

Novell Identity Manager

3

November 13, 2006

POLICY BUILDER AND DRIVER
CUSTOMIZATION GUIDE

www.novell.com



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/ 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

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

Novell Trademarks

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

eDirectory is a trademark of Novell, Inc.

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

Nsure is a trademark of Novell, Inc.

Third-Party Materials

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

Contents

About This Guide	11
1 Policies and Filters	13
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	37
2 Defining Policies By Using the Policy Builder with Designer	39
2.1 Policies	39
2.2 Policy Builder Tasks in Designer	40
2.2.1 Opening Policy Builder	40
2.2.2 Creating a Policy	44
2.2.3 Creating a Rule	52
2.2.4 Creating an Argument	61
2.2.5 Editing a Policy	71
2.2.6 Using Predefined Rules	74
2.2.7 Testing Policies with the Policy Simulator	105
2.2.8 Editing the DirXML Script	114
2.3 Regular Expressions	120
2.4 XPath 1.0 Expressions	121
2.5 Conditions	122
2.5.1 If Association	122
2.5.2 If Attribute	123
2.5.3 If Class Name	124
2.5.4 If Destination Attribute	125
2.5.5 If Destination DN	127
2.5.6 If Entitlement	127
2.5.7 If Global Configuration Value	128
2.5.8 If Local Variable	129
2.5.9 If Named Password	131
2.5.10 If Operation	132
2.5.11 If Operation Attribute	133
2.5.12 If Operation Property	135
2.5.13 If Password	136
2.5.14 If Source Attribute	136
2.5.15 If Source DN	137
2.5.16 If XPath Expression	138
2.6 Actions	139
2.6.1 Add Association	140
2.6.2 Add Destination Attribute Value	141
2.6.3 Add Destination Object	142
2.6.4 Add Source Attribute Value	144
2.6.5 Add Source Object	145
2.6.6 Append XML Element	146
2.6.7 Append XML Text	147

2.6.8	Break	148
2.6.9	Clear Destination Attribute Value	148
2.6.10	Clear Operation Property	149
2.6.11	Clear Source Attribute Value	149
2.6.12	Clear SSO Credential	150
2.6.13	Clone By XPath Expressions	151
2.6.14	Clone Operation Attribute	151
2.6.15	Delete Destination Object	152
2.6.16	Delete Source Object	153
2.6.17	Find Matching Object	153
2.6.18	For Each	155
2.6.19	Generate Event	156
2.6.20	Implement Entitlement	158
2.6.21	Move Destination Object	159
2.6.22	Move Source Object	160
2.6.23	Reformat Operation Attribute	161
2.6.24	Remove Association	162
2.6.25	Remove Destination Attribute Value	163
2.6.26	Remove Source Attribute Value	164
2.6.27	Rename Destination Object	165
2.6.28	Rename Operation Attribute	165
2.6.29	Rename Source Object	166
2.6.30	Send Email	166
2.6.31	Send Email From Template	168
2.6.32	Set Default Attribute Value	169
2.6.33	Set Destination Attribute Value	170
2.6.34	Set Destination Password	171
2.6.35	Set Local Variable	172
2.6.36	Set Operation Association	173
2.6.37	Set Operation Class Name	174
2.6.38	Set Operation Destination DN	174
2.6.39	Set Operation Property	175
2.6.40	Set Operation Source DN	176
2.6.41	Set Operation Template DN	176
2.6.42	Set Source Attribute Value	177
2.6.43	Set Source Password	178
2.6.44	Set SSO Credential	179
2.6.45	Set SSO Passphrase	179
2.6.46	Set XML Attribute	180
2.6.47	Status	181
2.6.48	Strip Operation Attribute	182
2.6.49	Strip XPath	182
2.6.50	Trace Message	183
2.6.51	Veto	184
2.6.52	Veto If Operational Attribute Not Available	185
2.7	Noun Tokens	186
2.7.1	Added Entitlement	186
2.7.2	Association	187
2.7.3	Attribute	187
2.7.4	Class Name	188
2.7.5	Destination Attribute	188
2.7.6	Destination DN	189
2.7.7	Destination Name	190
2.7.8	Entitlement	190
2.7.9	Global Configuration Value	191
2.7.10	Local Variable	191
2.7.11	Named Password	192
2.7.12	Operation	192

2.7.13	Operation Attribute	193
2.7.14	Operation Property	194
2.7.15	Password	194
2.7.16	Removed Attribute	194
2.7.17	Removed Entitlement	194
2.7.18	Source Attribute	195
2.7.19	Source DN	195
2.7.20	Source Name	196
2.7.21	Text	196
2.7.22	Unique Name	197
2.7.23	Unmatched Source DN	199
2.7.24	XPath	200
2.8	Verb Tokens	200
2.8.1	Escape Destination DN	200
2.8.2	Escape Source DN	201
2.8.3	Lower Case	201
2.8.4	Parse DN	202
2.8.5	Replace All	204
2.8.6	Replace First	205
2.8.7	Substring	206
2.8.8	Upper Case	207
2.9	Values	208
2.9.1	Comparison Modes	208

3 Defining Policies By Using the Policy Builder in iManager 211

3.1	Policies	211
3.2	Policy Builder Tasks in iManager	212
3.2.1	Opening The Policy Builder	212
3.2.2	Creating a Policy	212
3.2.3	Defining Individual Rules within a Policy	213
3.2.4	Defining Individual Arguments within a Rule	214
3.2.5	Modifying a Policy	222
3.2.6	Removing a Policy	222
3.2.7	Renaming a Policy	222
3.2.8	Deleting a Policy	223
3.2.9	Importing a Policy from an XML File	223
3.2.10	Exporting a Policy to an XML File	223
3.2.11	Creating a Policy Reference	223
3.2.12	Using Predefined Rules	224
3.3	Regular Expressions	244
3.4	XPath 1.0 Expressions	245
3.5	Conditions	246
3.5.1	If Association	246
3.5.2	If Attribute	247
3.5.3	If Class Name	248
3.5.4	If Destination Attribute	249
3.5.5	If Destination DN	250
3.5.6	If Entitlement	251
3.5.7	If Global Configuration Value	253
3.5.8	If Local Variable	254
3.5.9	If Named Password	256
3.5.10	If Operation	256
3.5.11	If Operation Attribute	258
3.5.12	If Operation Property	259
3.5.13	If Password	260
3.5.14	If Source Attribute	261

3.5.15	If Source DN	262
3.5.16	If XPath Expression	263
3.6	Actions	264
3.6.1	Add Association	265
3.6.2	Add Destination Attribute Value	266
3.6.3	Add Destination Object	267
3.6.4	Add Source Attribute Value	269
3.6.5	Add Source Object	269
3.6.6	Append XML Element	270
3.6.7	Append XML Text	271
3.6.8	Break	272
3.6.9	Clear Destination Attribute Value	272
3.6.10	Clear Operation Property	273
3.6.11	Clear SSO Credential	273
3.6.12	Clear Source Attribute Value	274
3.6.13	Clone By XPath Expression	274
3.6.14	Clone Operation Attribute	275
3.6.15	Delete Destination Object	276
3.6.16	Delete Source Object	276
3.6.17	Find Matching Object	276
3.6.18	For Each	278
3.6.19	Generate Event	279
3.6.20	Implement Entitlement	281
3.6.21	Move Destination Object	282
3.6.22	Move Source Object	283
3.6.23	Reformat Operation Attribute	284
3.6.24	Remove Association	284
3.6.25	Remove Destination Attribute Value	285
3.6.26	Remove Source Attribute Value	286
3.6.27	Rename Destination Object	287
3.6.28	Rename Operation Attribute	287
3.6.29	Rename Source Object	288
3.6.30	Send Email	288
3.6.31	Send Email from Template	289
3.6.32	Set Default Attribute Value	291
3.6.33	Set Destination Attribute Value	292
3.6.34	Set Destination Password	293
3.6.35	Set Local Variable	294
3.6.36	Set Operation Association	295
3.6.37	Set Operation Class Name	295
3.6.38	Set Operation Destination DN	295
3.6.39	Set Operation Property	296
3.6.40	Set Operation Source DN	296
3.6.41	Set Operation Template DN	297
3.6.42	Set Source Attribute Value	297
3.6.43	Set Source Password	298
3.6.44	Set SSO Credential	299
3.6.45	Set SSO Passphrase	299
3.6.46	Set XML Attribute	300
3.6.47	Status	301
3.6.48	Strip Operation Attribute	301
3.6.49	Strip XPath	302
3.6.50	Trace Message	302
3.6.51	Veto	303
3.6.52	Veto if Operation Attribute Not Available	304
3.7	Noun Tokens	305
3.7.1	Added Entitlement	305
3.7.2	Association	306

3.7.3	Attribute	306
3.7.4	Class Name	307
3.7.5	Destination Attribute	307
3.7.6	Destination DN	308
3.7.7	Destination Name	309
3.7.8	Entitlement	309
3.7.9	Global Configuration Value	310
3.7.10	Local Variable	310
3.7.11	Named Password	311
3.7.12	Operation	311
3.7.13	Operation Attribute	312
3.7.14	Operation Property	313
3.7.15	Password	313
3.7.16	Removed Attribute	313
3.7.17	Removed Entitlements	313
3.7.18	Source Attribute	314
3.7.19	Source DN	314
3.7.20	Source Name	315
3.7.21	Text	315
3.7.22	Unique Name	316
3.7.23	Unmatched Source DN	317
3.7.24	XPath	318
3.8	Verb Tokens	318
3.8.1	Escape Destination DN	319
3.8.2	Escape Source DN	319
3.8.3	Lower Case	319
3.8.4	Parse DN	320
3.8.5	Replace All	322
3.8.6	Replace First	323
3.8.7	Substring	324
3.8.8	Upper Case	325
3.9	Values	326
3.9.1	Comparison Modes	326

4 Novell Credential Provisioning Policies 327

4.1	Credential Provisioning Policies with Novell SecureLogin	327
4.2	Implementing Credential Provisioning Policies with Novell SecureLogin	329
4.2.1	Meeting Requirements for Credential Provisioning Policies with Novell SecureLogin	329
4.2.2	Extending LDAP Schema for Novell SecureLogin	330
4.2.3	Determining Deployment Configuration Parameters for Novell SecureLogin	330
4.2.4	Creating a Repository Object for Novell SecureLogin	333
4.2.5	Creating an Application Object for Novell SecureLogin	339
4.2.6	Configuring Credential Provisioning Policies for Novell SecureLogin	345
4.3	Credential Provisioning Policies with Novell SecretStore	349
4.4	Implementing Credential Provisioning Policies with SecretStore	351
4.4.1	Meeting Requirements for Credential Provisioning Policies with Novell SecretStore	352
4.4.2	Determining Deployment Configuration Parameters for Novell SecretStore	352
4.4.3	Creating a Repository Object for Novell SecretStore	355
4.4.4	Creating an Application Object for Novell SecretStore	361
4.4.5	Configuring Credential Provisioning Policies for Novell SecretStore	368

5 Defining Policies using XSLT Style Sheets 373

5.1	Managing XSLT Style Sheets in Designer	373
-----	--	-----

5.1.1	Adding an XSLT Policy in Designer	373
5.2	Managing XSLT Style Sheets in iManager	375
5.2.1	Adding an XSLT Policy in iManager	375
5.3	Starting with an Identity Transformation	376
5.4	Using the Parameters that Identity Manager Passes	377
5.5	Using Extension Functions	379
5.6	Creating a Password Example: Creation Policy	380
5.7	Creating an eDirectory User Example: Creation Policy	381
6	Managing Filters	387
6.1	Filter Tasks in Designer	387
6.1.1	Accessing the Filter Editor	387
6.1.2	Editing the Filter	390
6.1.3	Testing Filters	394
6.1.4	Viewing the Filter XML Source	400
6.1.5	Additional Filter Options	406
6.2	Filter Tasks in iManager	408
6.2.1	Accessing the Filter	408
6.2.2	Editing the Filter	408
7	Managing Schema Mapping Policies	413
7.1	Schema Mapping Policy Tasks in Designer	413
7.1.1	Accessing the Schema Map Editor	413
7.1.2	Editing a Schema Mapping Policy	417
7.1.3	Testing Schema Mapping Policies	420
7.1.4	Accessing the Schema Mapping Policy XML	426
7.1.5	Additional Schema Map Policy Options	432
7.2	Schema Mapping Policy Tasks in iManager	436
7.2.1	Accessing Schema Mapping Policies	436
7.2.2	Editing the Schema Mapping Policy	436
A	Documentation Update	441
A.1	March 26, 2007	441
A.1.1	Introduction to Policies	441
A.2	October 3, 2006	441
A.2.1	Defining Policies By Using the Policy Builder with Designer	441
A.2.2	Defining Policies By Using the Policy Builder with iManager	442
A.3	September 8, 2006	442
A.3.1	Implementing Credential Provisioning Policies with Novell SecureLogin	442
A.3.2	Configuring Credential Provisioning Policies for Novell SecureLogin	442
A.3.3	Implementing Credential Provisioning Policies with Novell SecretStore	443
A.3.4	Configuring Credential Provisioning Policies for Novell SecretStore	443
A.4	July 31, 2006	443
A.4.1	Introduction to Policies	443

About This Guide

Novell® Identity Manager 3.0.1 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 3.0.1.

- ♦ [Chapter 1, “Policies and Filters,” on page 13](#)
- ♦ [Chapter 2, “Defining Policies By Using the Policy Builder with Designer,” on page 39](#)
- ♦ [Chapter 3, “Defining Policies By Using the Policy Builder in iManager,” on page 211](#)
- ♦ [Chapter 5, “Defining Policies using XSLT Style Sheets,” on page 373](#)
- ♦ [Chapter 6, “Managing Filters,” on page 387](#)
- ♦ [Chapter 7, “Managing Schema Mapping Policies,” on page 413](#)

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

Policies and Filters

1

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 39](#) or [Chapter 3, “Defining Policies By Using the Policy Builder in iManager,” on page 211](#) to learn how to use the Policy Builder.

1.1.1 Terminology Changes from Earlier Versions

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 caused confusion in circumstances when the context was 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, and 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

Concept	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 3.0.1.

Table 1-2 Terminology Changes from Identity Manager 2.x to Identity Manager 3.x

Concept	Identity Manager 2.x Terminology	Identity Manager 3.x 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 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](#)

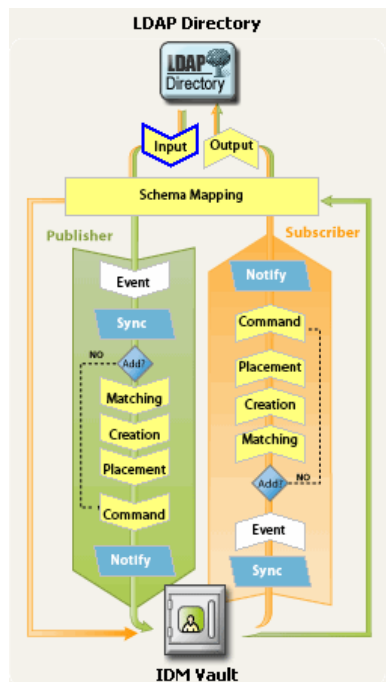
1.2.1 Policies

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” on page 16
- ◆ “Matching Policies” on page 19
- ◆ “Creation Policy” on page 20
- ◆ “Placement Policy” on page 23
- ◆ “Command Transformation Policy” on page 26
- ◆ “Schema Mapping Policy” on page 29
- ◆ “Output Transformation Policy” on page 31
- ◆ “Input Transformation Policy” on page 34

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>

```

```

        </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>

```

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 datastore. 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>
  </rule>
</policy>

```

```

        <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>

```

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>

```

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>
  <rule>
    <description>Conditionally veto guys named "Fred"</
description>
    <conditions>
      <and>
        <if-global-variable name="no-fred"
op="equal">true</if-global-variable>
        <if-op-attr name="Given Name"
op="equal">Fred</if-op-attr>
      </and>
    </conditions>
  </rule>
</policy>
```

```

        </conditions>
        <actions>
            <do-status level="warning">
                <arg-string>
                    <token-text xml:space="preserve"
xmlns:xml="http://www.w3.org/XML/1998/namespace">Vetoed "Fred"</token-
text>
                </arg-string>
            </do-status>
        </do-veto/>
    </actions>
</rule>
</policy>

```

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>

```

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 7, “Managing Schema Mapping Policies,” on page 413](#) 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 39](#) and [Chapter 3, “Defining Policies By Using the Policy Builder in iManager,” on page 211](#), 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 5, "Defining Policies using XSLT Style Sheets," on page 373](#) 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 the [Novell Support Web site \(http://support.novell.com/patches.html\)](http://support.novell.com/patches.html). To download the policies:

- 1 At the [Novell Support Web site \(http://support.novell.com/patches.html\)](http://support.novell.com/patches.html) select *View the minimum patch list*.
- 2 Browse to and select Identity Manager in the *Product or Technology* field, then click *search*.
- 3 Browse to and select the desired policy.
[Table 1-3](#) contains a list of the policies available for download.
- 4 Select *proceed to download*, to download the policy.
- 5 Click *download* by the file name.
- 6 Click *Save*, then browse to and select a location to save the file.
- 7 Click *Save*, then click *Close*.
- 8 Extract the file, then read the `How_To_Install.rtf` file for installation instructions.

Table 1-3 Downloadable Policies

Name	File Name
Policy to Place by Surname	placebyname.tgz
Policy: Reset value of the email attribute	pushback.tgz
Policy to enforce the presence of attributes	requiredattrs.tgz
Policy: Create email from GivenName & Surname	setemailname.tgz
Policy: Create FullName from GivenName, Surname	synthfullname.tgz

Name	File Name
Policy: Convert First/Last name to uppercase	uppercasenames.tgz
Policy to add user to group based on Title	addcreategroups.tgz
Policy: Assign template to user based on title	assigntemplate.tgz
Disable user account and move when terminated	dismvonterm.tgz
Policy to filter events	filterby.tgz
Govern Groups for user based on the title attribute	groupchange.tgz

To use Designer to import the files, see [“Importing a Policy From an XML File” on page 60](#). To use iManager to import the files, see [Section 3.2.9, “Importing a Policy from an XML File,” on page 223](#).

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 6, “Managing Filters,” on page 387](#) for information on defining filters.

Defining Policies By Using the Policy Builder with Designer

2

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 39](#)
- ◆ [Section 2.2, “Policy Builder Tasks in Designer,” on page 40](#)

This section also contains the following detailed reference sections:

- ◆ [Section 2.3, “Regular Expressions,” on page 120](#)
- ◆ [Section 2.4, “XPath 1.0 Expressions,” on page 121](#)
- ◆ [Section 2.5, “Conditions,” on page 122](#)
- ◆ [Section 2.6, “Actions,” on page 139](#)
- ◆ [Section 2.7, “Noun Tokens,” on page 186](#)
- ◆ [Section 2.8, “Verb Tokens,” on page 200](#)

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 122](#)) that must be met before a defined action (see [“Actions” on page 139](#)) 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 186](#)) and verbs (see [“Verb Tokens” on page 200](#)).
 - ◆ 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 120](#)) and XPath 1.0 expressions (see [“XPath 1.0 Expressions” on page 121](#)) 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.
- ◆ An operation is any element in the XDS document 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 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).
- ◆ An operation usually represents an event, a command, or a status.

- ◆ 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.
- ◆ A policy can also get additional context from outside of the document and cause side effects that are not reflected in the result document.

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 40](#)
- ◆ [Section 2.2.2, “Creating a Policy,” on page 44](#)
- ◆ [Section 2.2.3, “Creating a Rule,” on page 52](#)
- ◆ [Section 2.2.4, “Creating an Argument,” on page 61](#)
- ◆ [Section 2.2.5, “Editing a Policy,” on page 71](#)
- ◆ [Section 2.2.6, “Using Predefined Rules,” on page 74](#)
- ◆ [Section 2.2.7, “Testing Policies with the Policy Simulator,” on page 105](#)
- ◆ [Section 2.2.8, “Editing the DirXML Script,” on page 114](#)

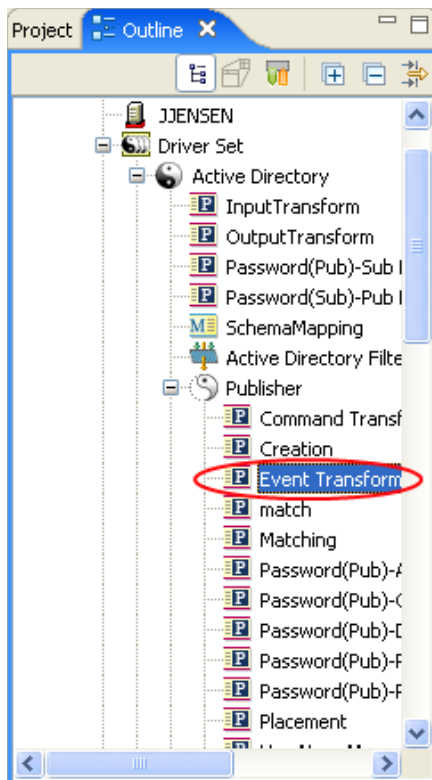
2.2.1 Opening Policy Builder

The Policy Builder can be opened from the Model Outline view, from the Policy Flow view, or from a policy set.

Model Outline View

- 1 Open a project in Designer.
- 2 Click the *Outline* tab > select the *Show Model Outline* icon.

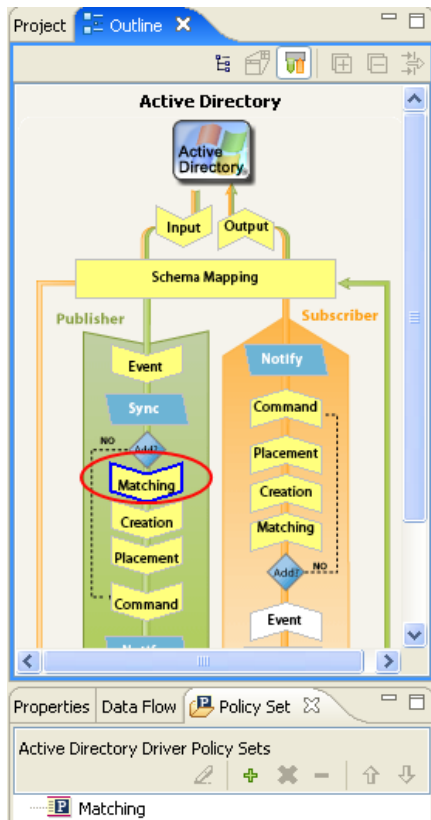
- 3 Double-click a policy listed in the Model Outline view or right-click and select *Edit*.



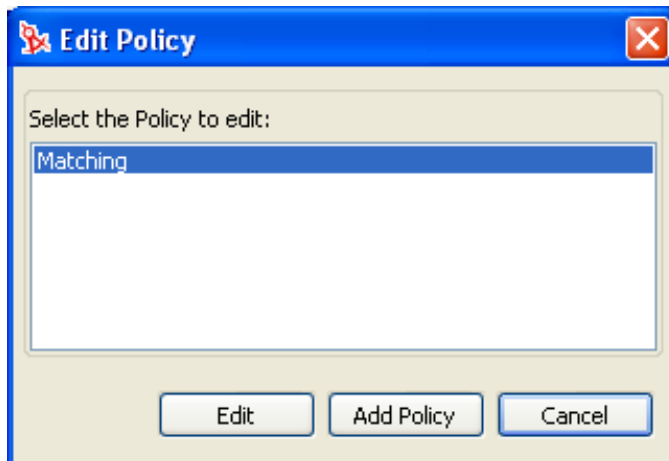
Policy Flow View

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

- 3 Right-click a policy (for example, the Matching policy) in the Policy Flow view, then select *Edit Policy*.

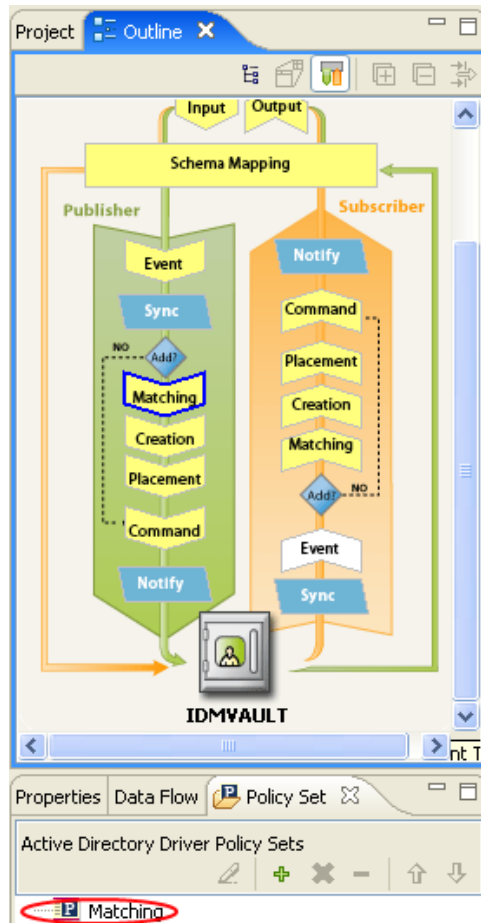


- 4 You can also double-click the Matching policy in the Policy Flow.
- 5 Select the policy, then click *Edit*.



Policy Set

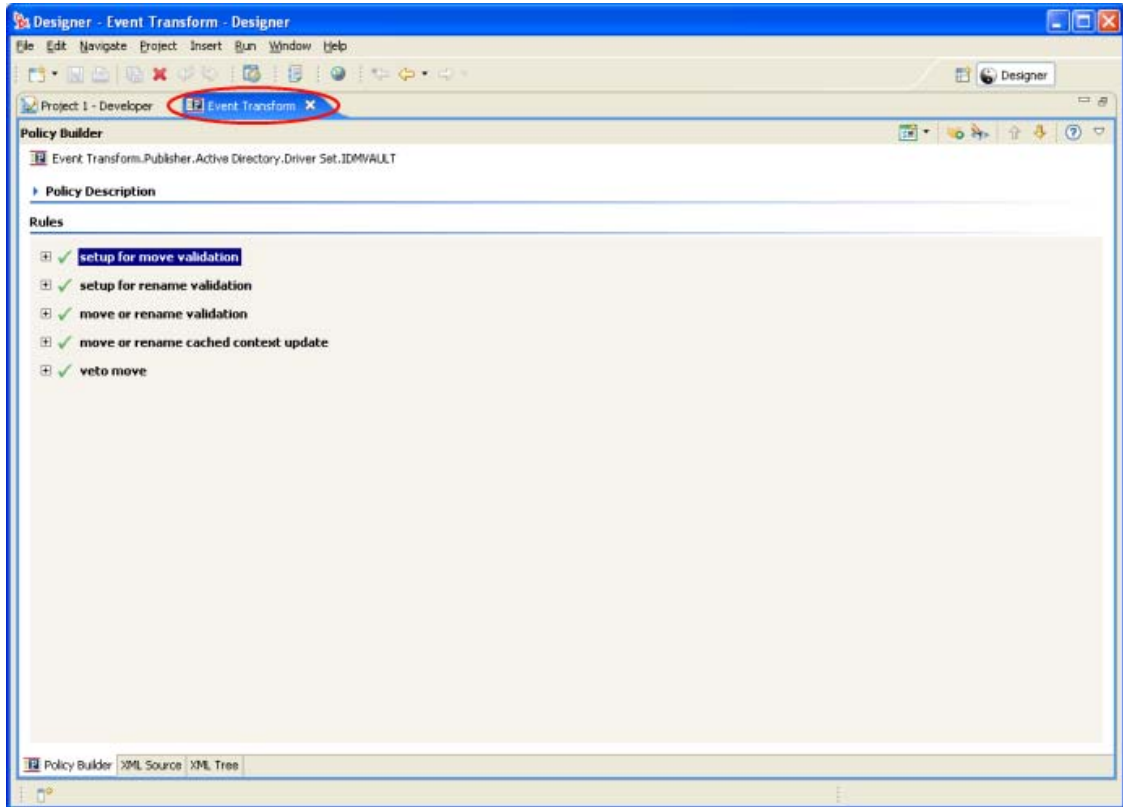
- 1 Right-click the policy in the policy set, then click *Edit*.



- 2 You can also 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.

Figure 2-1 Policy Builder Full Screen



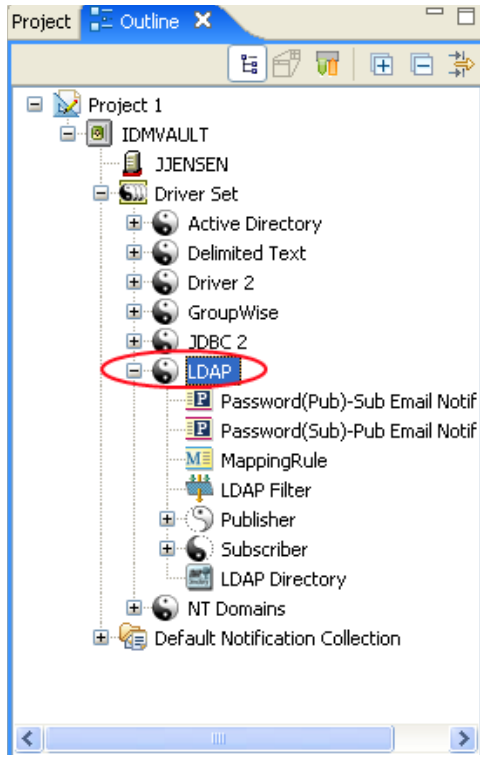
2.2.2 Creating a Policy

A policy sends data to the connected systems. A policy is created through the policy set.

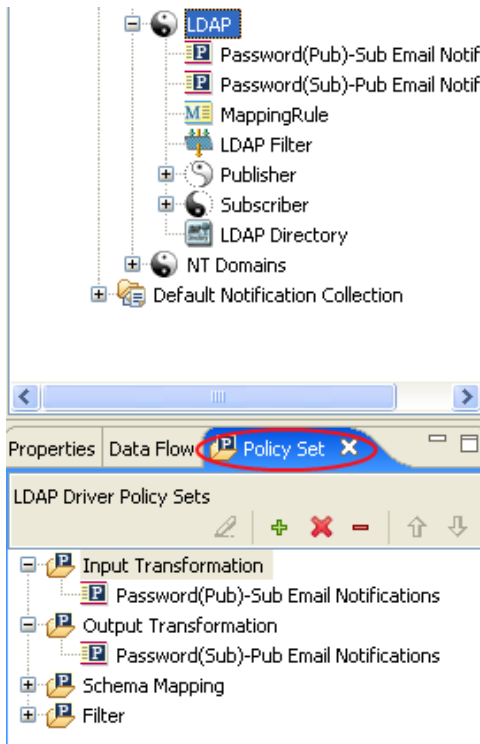
- ◆ [“Accessing the Policy Set” on page 45](#)
- ◆ [“Using the Policy Set” on page 46](#)
- ◆ [“Using the Add Policy Wizard” on page 48](#)

Accessing the Policy Set

- 1 Select a driver object from the *Outline* view in an open project.

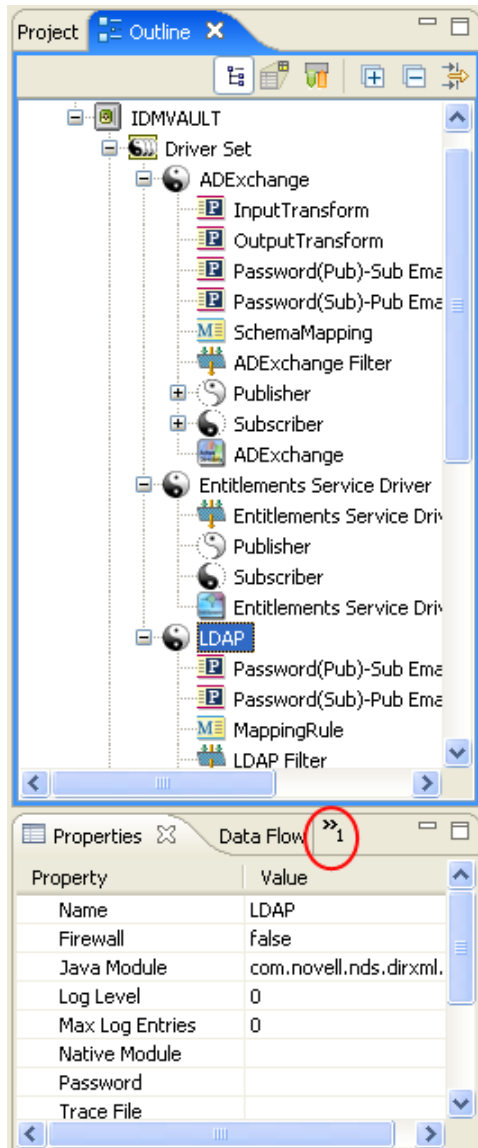


- 2 Select the *Policy Set* tab.

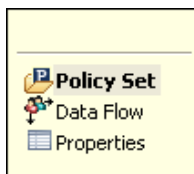


If the *Policy Set* tab is not shown:

- 1 Click the double arrow.



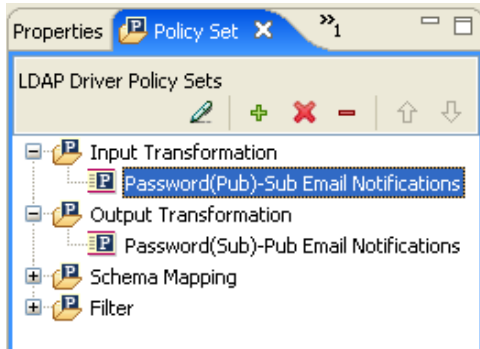
- 2 Select *Policy Set*.



Using the 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, including, 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 the item you have selected. The different icons are described below.

Table 2-1 Policy Set Toolbar

Operation	Description
Edit a policy	Launches the Policy Builder.
Create or add a new policy to the Policy Set	Launches the Add Policy Wizard.
Remove and delete the selected policy	Deletes the policy from the project.
Remove the selected policy from the Policy Set, do not delete	Removes the policy from the selected policy set object but doesn't delete the policy.
Move the policy up the policy chain	Moves the policy up in the processing order.
Move the policy down the policy chain	Moves the policy down in the processing order.

Keyboard Support

You can move through the policy set with keystrokes as well as using the mouse. The supported keystrokes are listed below.

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.
Delete	Deletes the policy from the project.

Keystroke	Description
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.

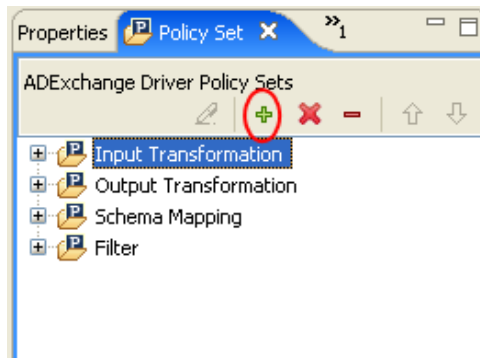
Using the Add Policy Wizard

The Add Policy Wizard launches when you click the *Create or add a new policy to the Policy Set* icon in the toolbar. The Add Policy Wizard enables you to do the following:

- ♦ “Creating a Policy” on page 48
- ♦ “Copying a Policy” on page 50
- ♦ “Linking to a Policy” on page 51

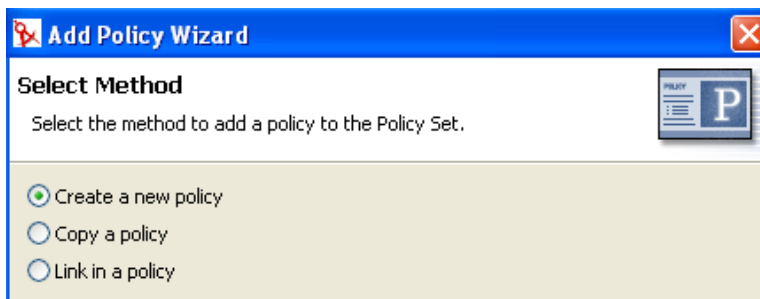
To launch the Add Policy Wizard:

- 1 Select a driver in the *Outline* view.
- 2 Select a policy set item in the policy set, then click the *Create or add a new policy to the Policy Set* icon in the toolbar.

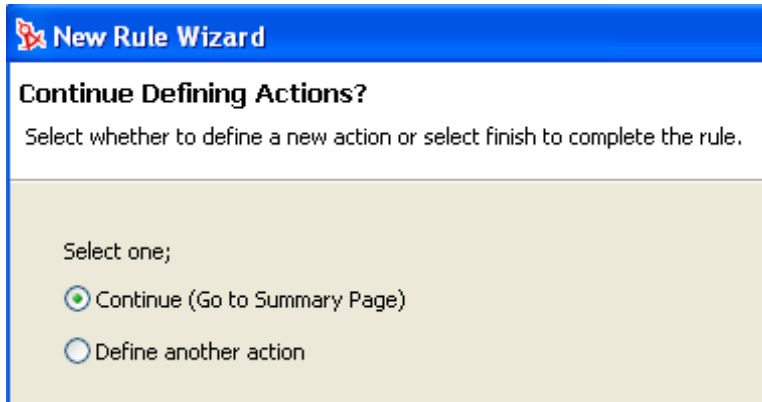


Creating a Policy

- 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.

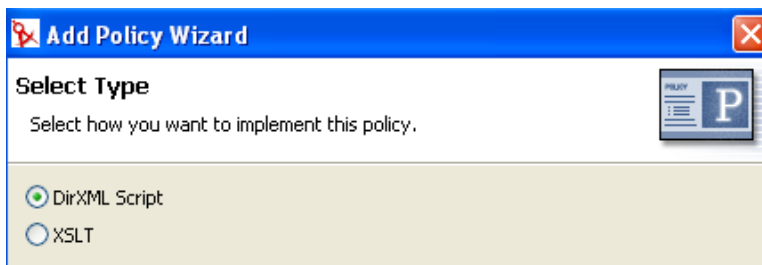
This decision depends on how you want to organize the policies. By default, policies are placed under the container object that is selected in the *Outline* tab when the Add Policy Wizard is launched.

For example, if you move to a Publisher object in the *Outline* tab 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.



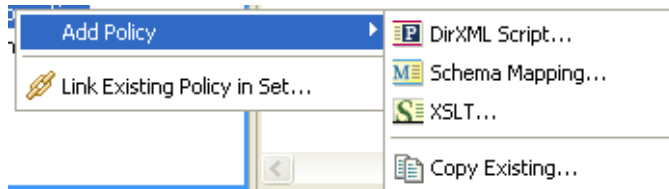
- 5 Click *Finish*.

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

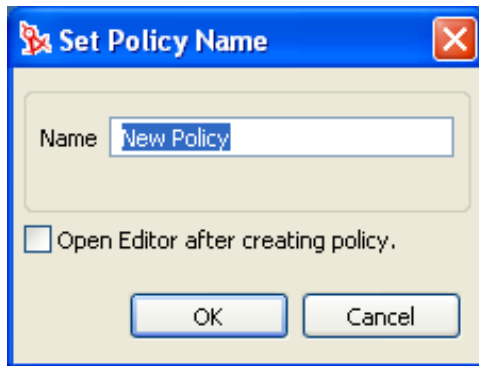
You can also add a policy by right-clicking a policy set.

- 1 Right-click a policy set (for example, Input Transformation Set).

- 2 Select *Add Policy*.
- 3 Select how to implement the policy: *DirXML Script*, *Schema Mapping*, *XSLT* or *Copy Existing*.



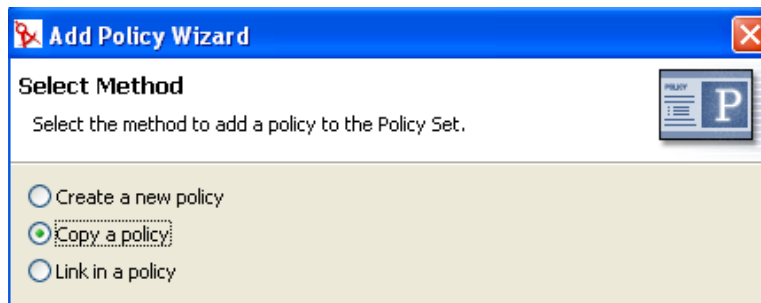
- 4 Name the policy.



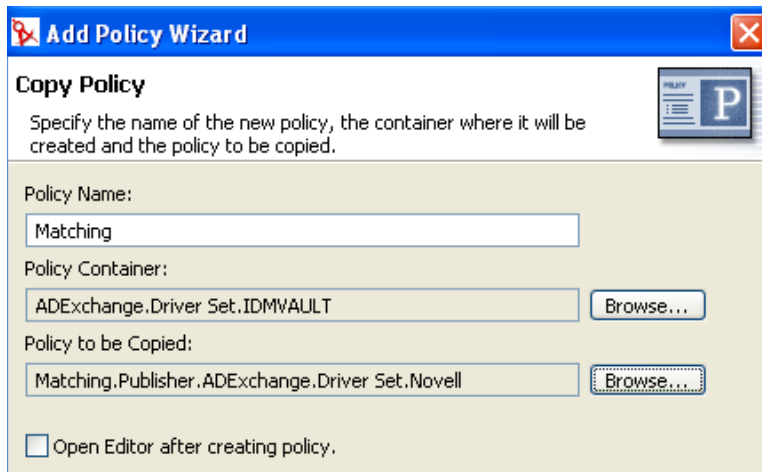
- 5 Click *Open Editor after creating policy*.
- 6 Click *OK*.

Copying a Policy

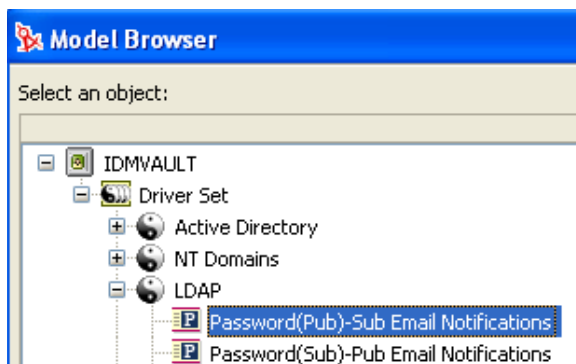
- 1 In the Add Policy Wizard, select *Copy a policy*, then click *Next*.



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.

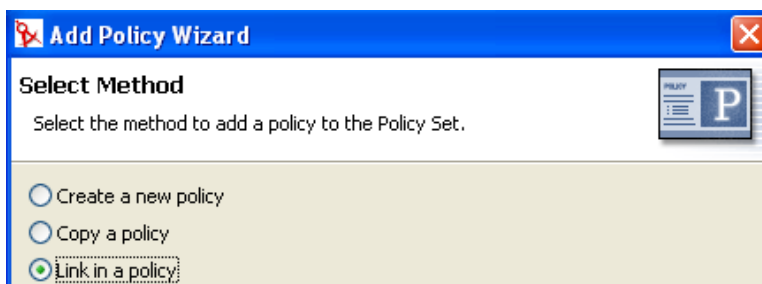


4 Browse to and select the policy you want to copy, then click *OK*.

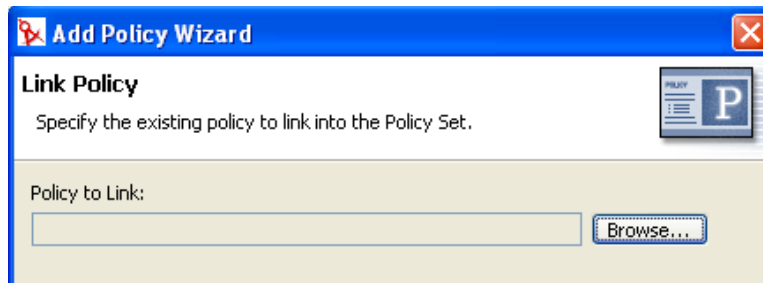
5 Click *Finish* to make a copy of the selected policy.

Linking to a Policy

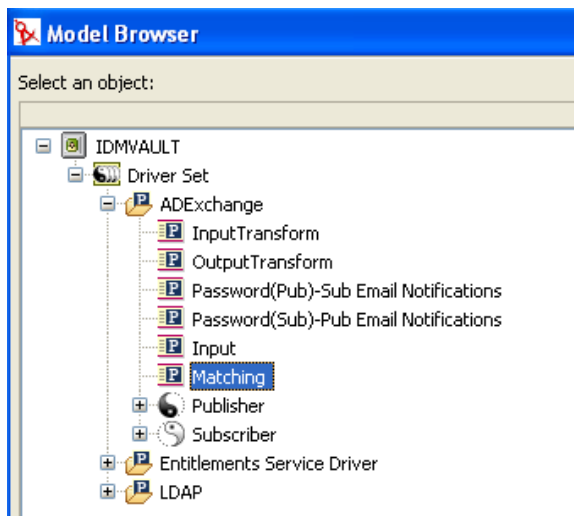
1 In the Add Policy Wizard, select *Link in a policy*, then click *Next*.



2 Click *Browse* to launch the model browser.



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.

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

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

2.2.3 Creating a Rule

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

Rules can be created in four different ways:

- ◆ "Creating a New Rule" on page 53
- ◆ "Using Predefined Rules" on page 57
- ◆ "Including an Existing Rule" on page 58
- ◆ "Importing a Policy From an XML File" on page 60

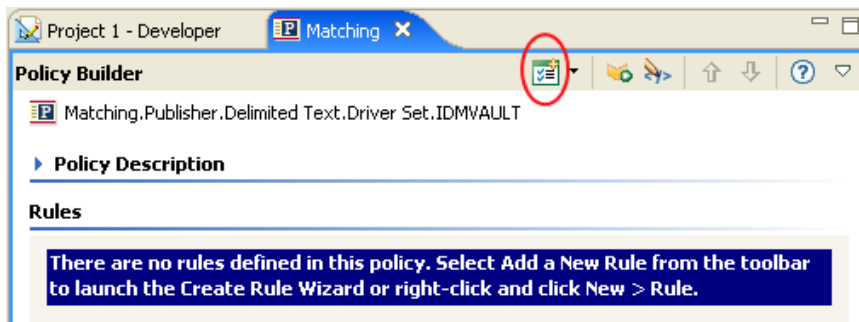
Creating a New Rule

When you create a rule, you create condition groups, conditions, and actions. Each rule is composed of conditions, actions, and arguments. For more information, click the Help icon (?) when creating each item. The help files contain a definition and an example of the item being used.

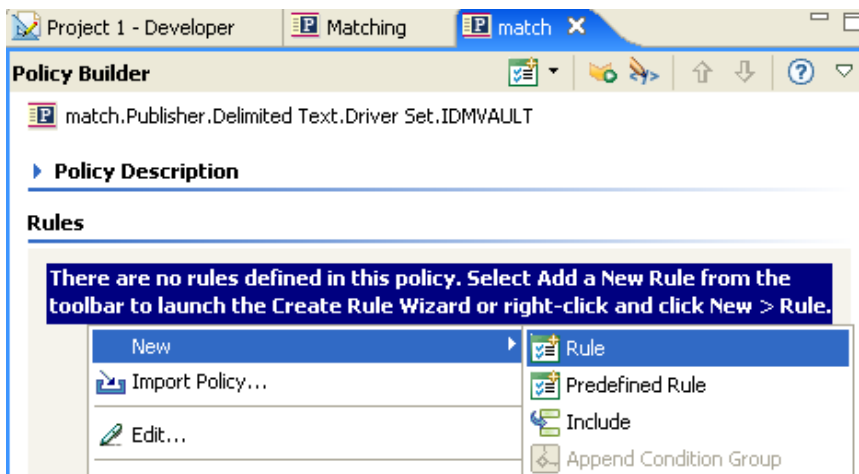
- ♦ “Creating a Rule” on page 53
- ♦ “Creating a Conditional Group” on page 56
- ♦ “Creating a Condition” on page 57
- ♦ “Creating an Action” on page 57

Creating a Rule

- 1 From the Policy Builder toolbar, select *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*.

The screenshot shows the 'New Rule Wizard' dialog box with a blue header. Below the header, the title 'Name and Describe Rule' is displayed. A sub-header explains: 'The rule and description display on the rule in the Rule Builder editor. Both can be edited by double-clicking the rule name in Rule Builder.' Below this text are two input fields: 'Name' with a placeholder '<Enter Name>' and 'Description' with a placeholder '<Enter Description and Comments>'.

- 3 Select the condition structure (*OR Conditions, AND Groups* or *AND Conditions, OR Groups*) then click *Next*.

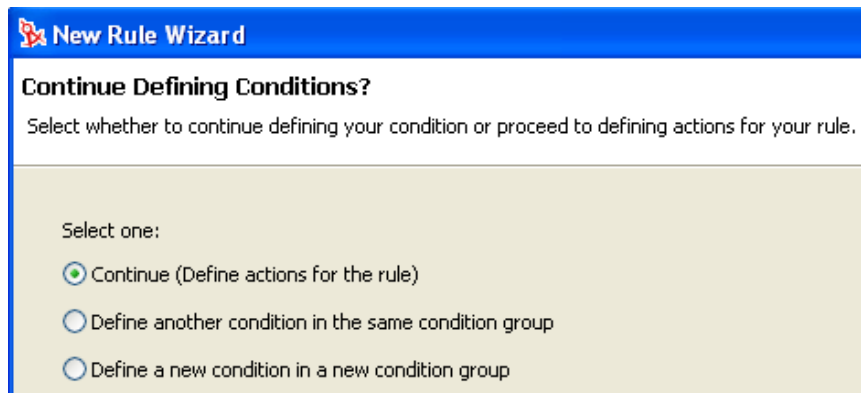
The screenshot shows the 'New Rule Wizard' dialog box with a blue header. Below the header, the title 'Select the Condition Structure' is displayed. A sub-header explains: 'Condition structures define the logic of condition groups.' Below this text are two radio button options: 'OR Conditions, AND Groups' (unselected) and 'AND Conditions, OR Groups' (selected).

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

The screenshot shows the 'New Rule Wizard' dialog box with a blue header. Below the header, the title 'Define the Condition' is displayed. A sub-header explains: '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.' Below this text are three input fields: 'Condition' with a dropdown menu showing 'attribute', 'Name *' with a text input field containing 'Given Name' and a magnifying glass icon, and 'Operator *' with a dropdown menu showing 'not available'. A help icon (?) is visible in the top right corner of the main content area.

Click the Help icon (?) for information about each condition you can create.

- 5 You can define an additional condition or condition group at this point. For this example, there is only one condition. Select *Continue*, then click *Next*.



New Rule Wizard

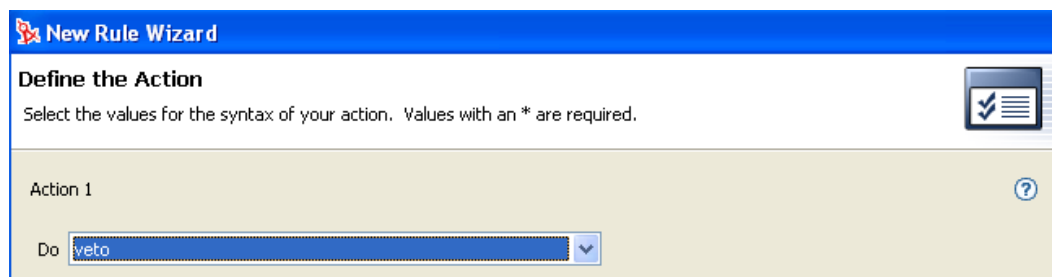
Continue Defining Conditions?

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

Select one:

- Continue (Define actions for the rule)
- Define another condition in the same condition group
- Define a new condition in a new condition group

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



New Rule Wizard

Define the Action

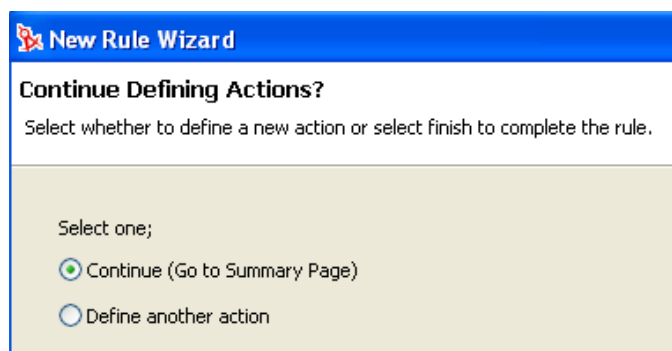
Select the values for the syntax of your action. Values with an * are required.

Action 1 ?

Do

Click the Help icon ? for information about each action you can create.

- 7 You can define additional actions at this point. For this example, there is only one action. Select *Continue*, then click *Next*.



New Rule Wizard

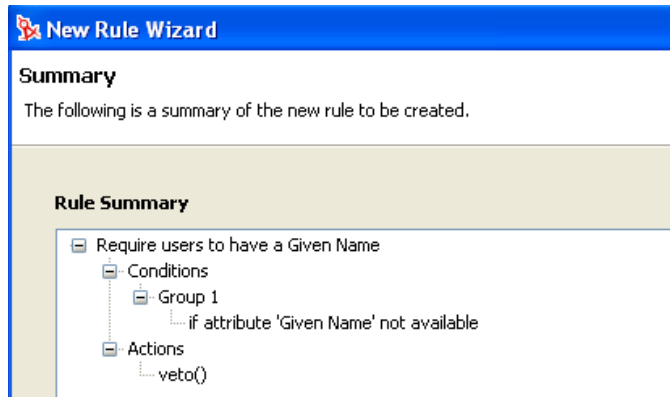
Continue Defining Actions?

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

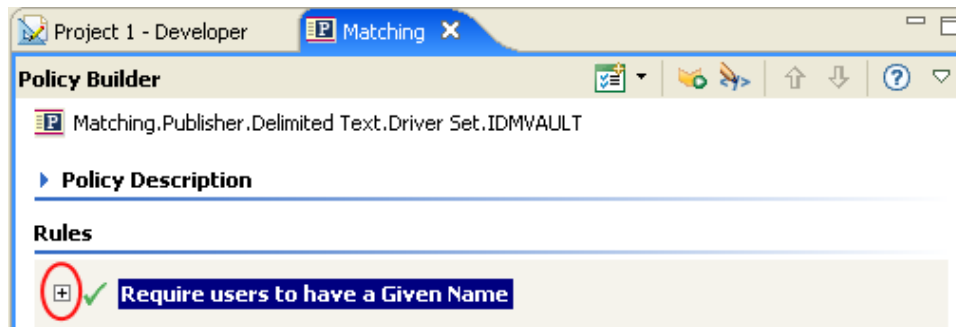
Select one;

- Continue (Go to Summary Page)
- Define another action

- 8 The summary page displays the rule that was created. Click *Finish* to complete the creation of the rule.

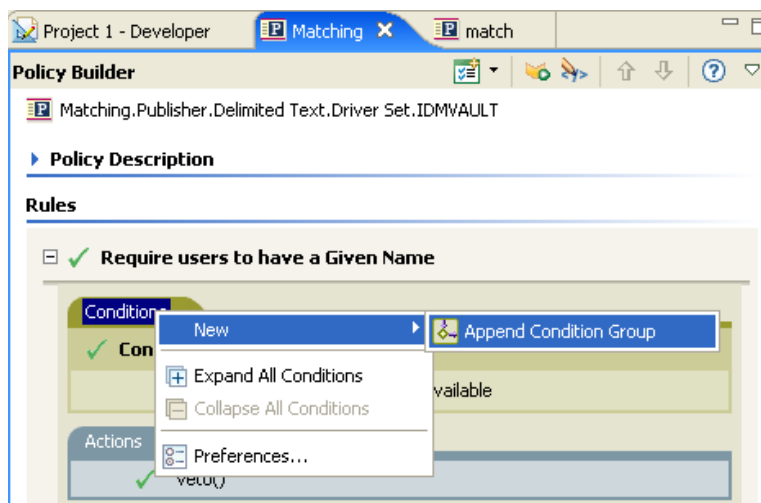


You can expand or collapse the view of the rule by clicking the plus or minus sign.



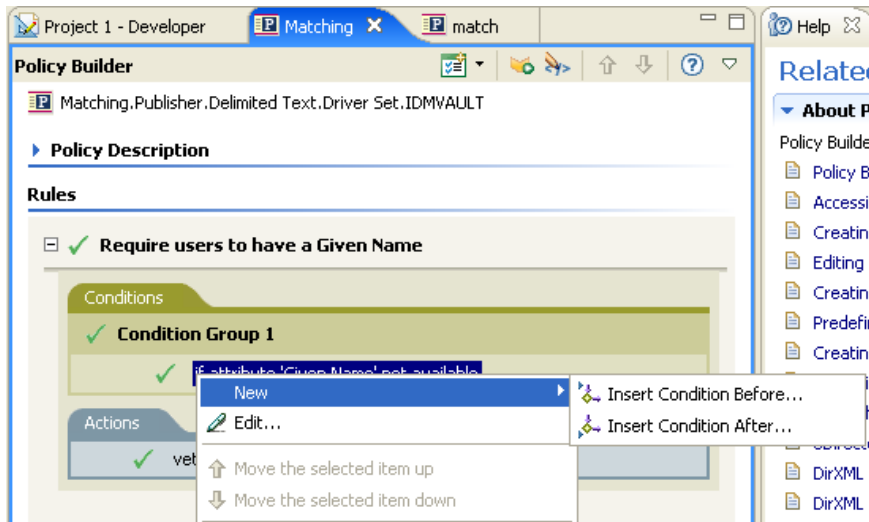
Creating a Conditional Group

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



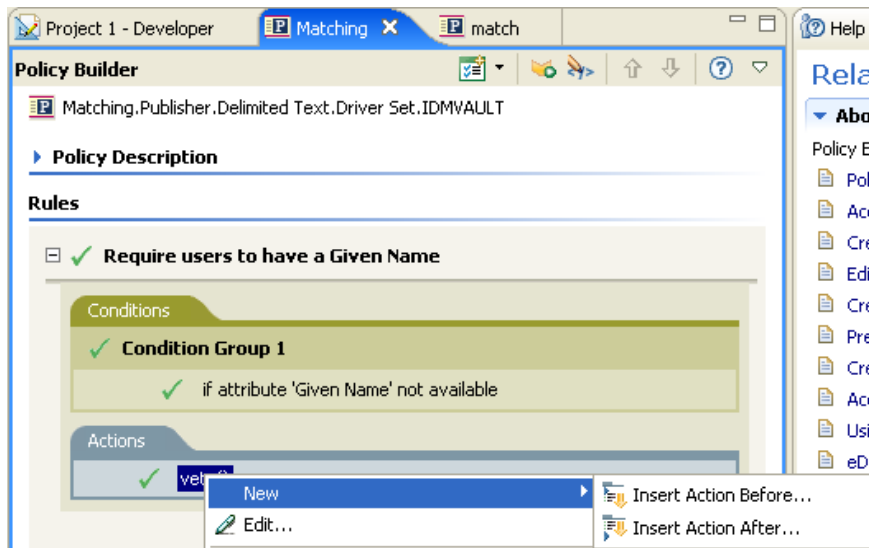
Creating a Condition

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



Creating an Action

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

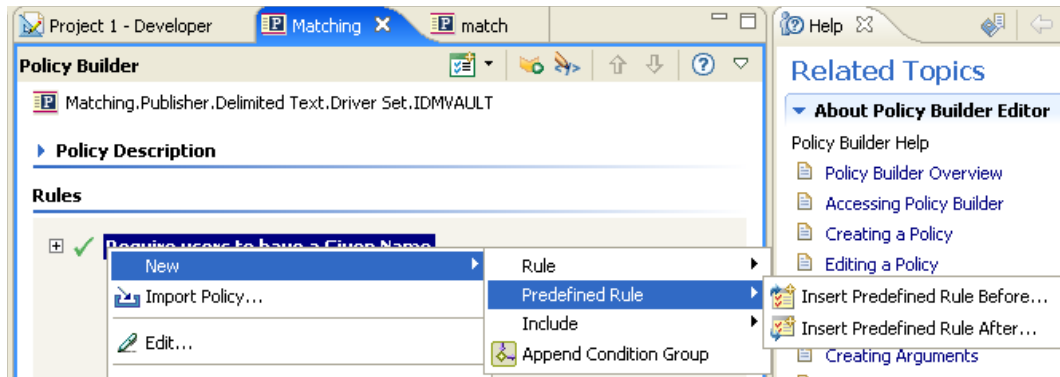


Using Predefined Rules

Designer includes a list of predefined rules. You can import and use these rules as well as create your own rules.

- 1 Right-click in the Policy Builder and select *New > Predefine Rules > Insert Predefined Rule Before* or *Insert Predefined Rule After*.

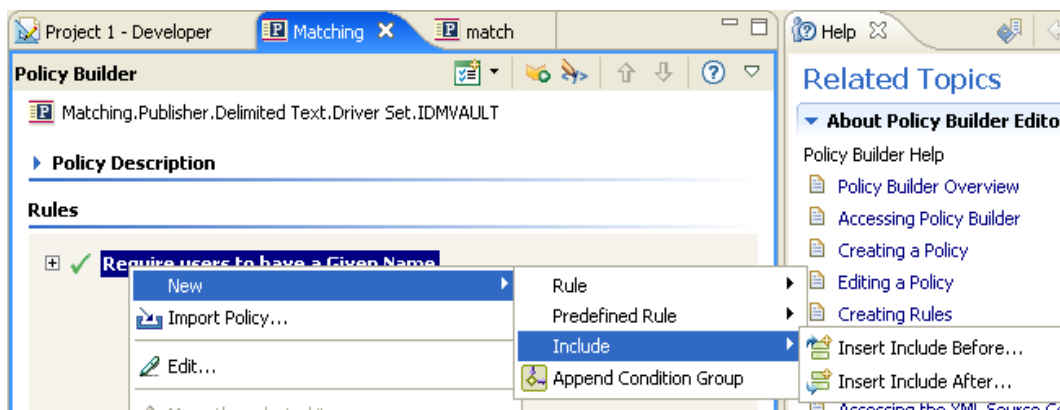
See [Section 2.2.6, “Using Predefined Rules,”](#) on page 74 for more information.



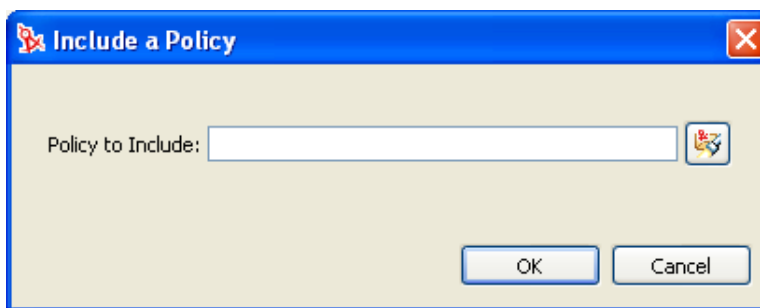
Including an Existing Rule

Designer allows you to include the rules from another policy.

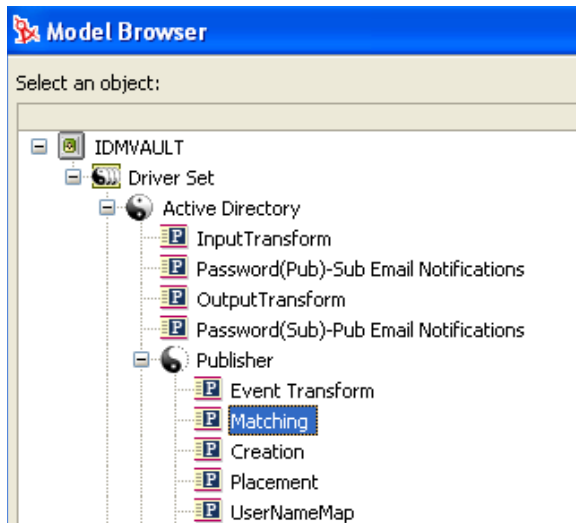
- 1 Right-click in the Policy Builder and click *New > Include > Insert Include Before* or *Insert Include After*.



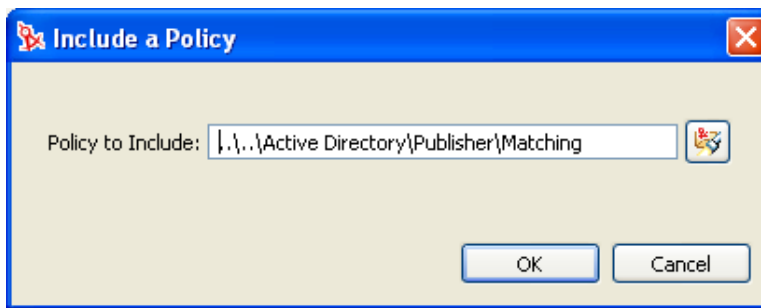
- 2 Click the Browse icon.



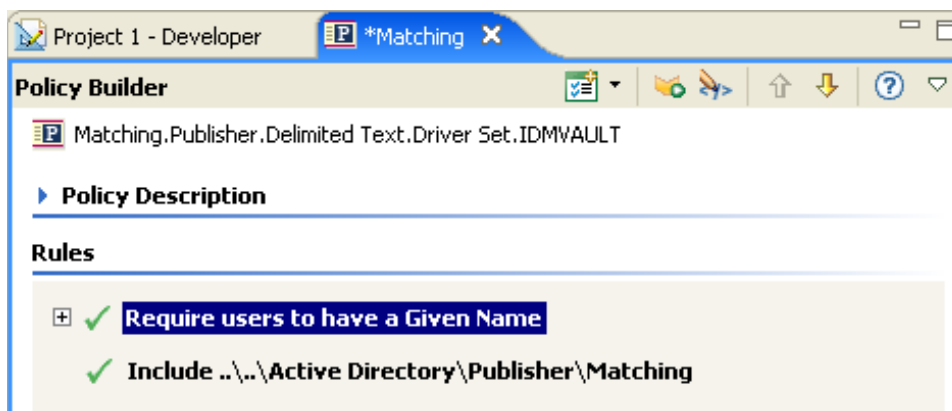
- 3 Browse to the policy you want to include, then click *OK*.



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



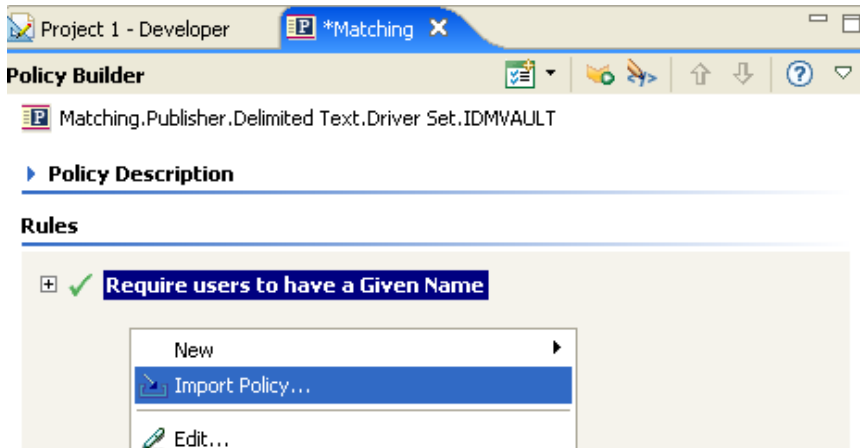
The rule is a link to the original rule. You cannot edit the rule in this location. Access the original rule to make changes.



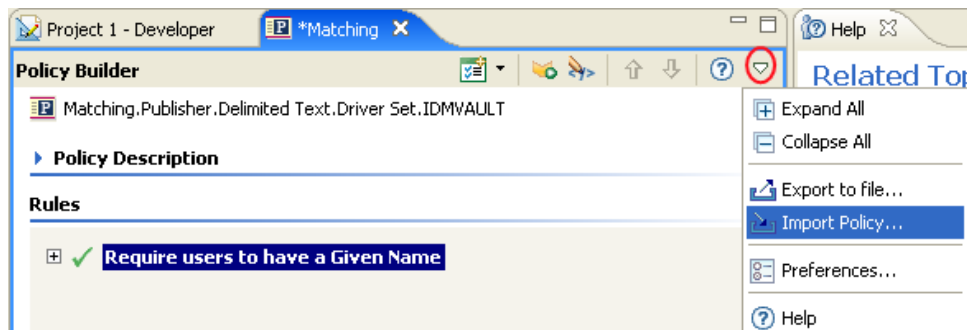
Importing a Policy From an XML File

Rules and policies can be saved as XML files. If you have a file that contains a rule or a policy you want to use, the Policy Builder allows you to import the file.

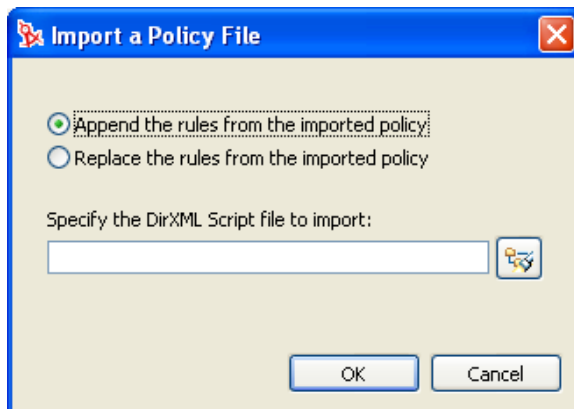
- 1 In the Policy Builder, right-click and select *Import Policy*.



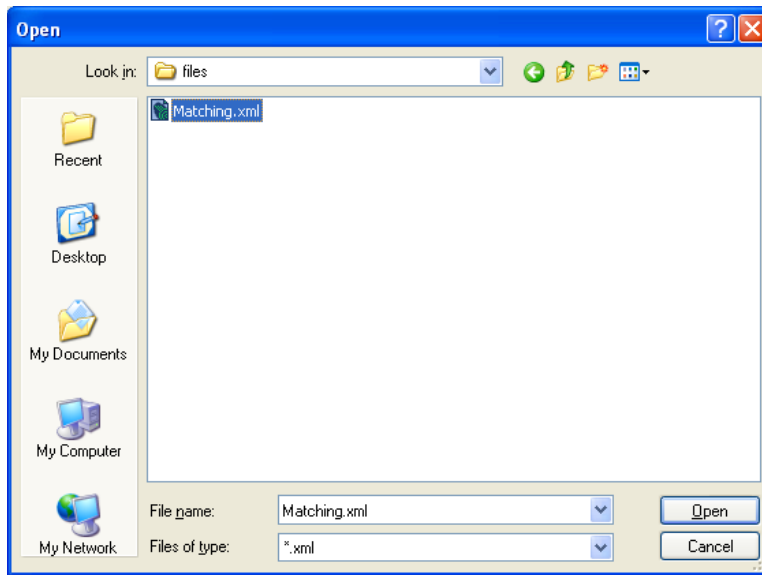
You can also select the Import Policy icon from the drop-down list in the toolbar.



- 2 Select one of the two options: *Append the rules from the imported policy* or *Replace the rules from the imported policy*.



3 Click the browse icon and select the file that contains the DirXML 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 64](#).

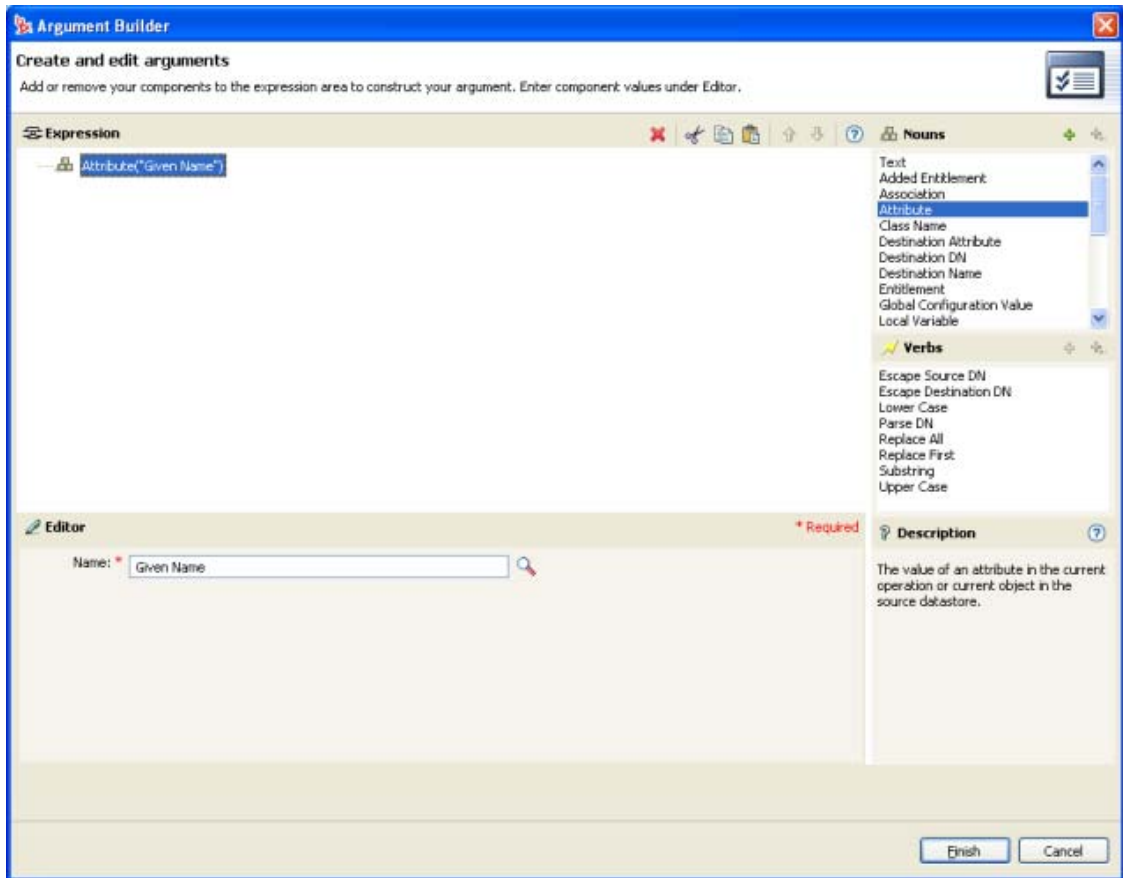
Arguments are dynamically used by actions and are derived from tokens that are expanded at run time.

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.

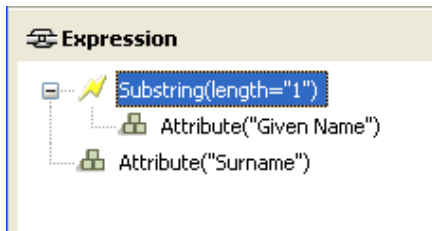
For example, if you want the argument set to an attribute value, you select the attribute noun, then select the attribute name:

Figure 2-2 *Argument Builder*



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

Figure 2-3 *Expression*



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


See [“Noun Tokens” on page 186](#) and [“Verb Tokens” on page 200](#) 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:

- ◆ “Actions Builder” on page 63
- ◆ “Argument Builder” on page 64
- ◆ “Match Attribute Builder” on page 65
- ◆ “Action Argument Component Builder” on page 66
- ◆ “Argument Value List Builder” on page 67
- ◆ “Named String Builder” on page 68
- ◆ “Condition Argument Component Builder” on page 69
- ◆ “Pattern String Builder” on page 70

The information below describes how to access each Builder.

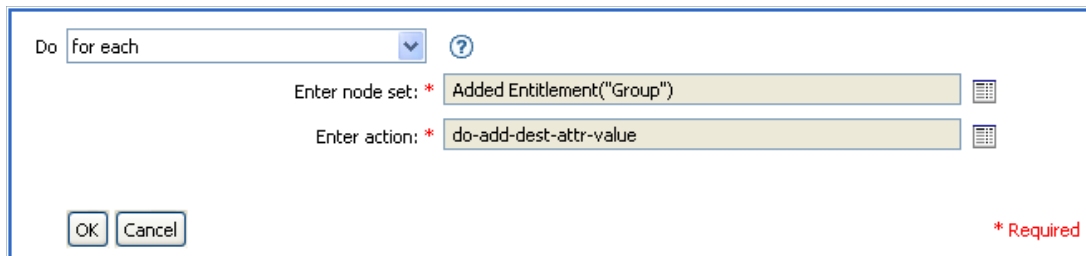
Actions Builder

To launch the Actions Builder, select one of following two actions, then click the *Edit the arguments* icon .

- ◆ For Each
- ◆ Implement Entitlement

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 2-4 For Each Action



The screenshot shows a dialog box for configuring the 'For Each' action. It features a dropdown menu with 'for each' selected, a help icon, and two required input fields: 'Enter node set: * Added Entitlement("Group")' and 'Enter action: * do-add-dest-attr-value'. Both fields have grid icons to the right. At the bottom left are 'OK' and 'Cancel' buttons, and at the bottom right is a red asterisk followed by the text '* Required'.

To define the action of the add destination attribute value, click the icon that launches the Actions Builder. In the 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 2-5 Argument Action Builder

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()

OK Cancel

* Required

Argument Builder

To launch the Argument Builder, select one of the following actions, then click the *Edit the Arguments* icon.

- ◆ “Add Association” on page 140
- ◆ “Add Destination Attribute Value” on page 141
- ◆ “Add Destination Object” on page 142
- ◆ “Add Source Attribute Value” on page 144
- ◆ “Append XML Text” on page 147
- ◆ “Clear Destination Attribute Value” on page 148 (when the selected object is DN or Association)
- ◆ “Clear Source Attribute Value” on page 149 (when the selected object is DN or Association)
- ◆ “Delete Destination Object” on page 152 (when the selected object is DN or Association)
- ◆ “Delete Source Object” on page 153 (when the selected object is DN or Association)
- ◆ “Find Matching Object” on page 153
- ◆ “For Each” on page 155
- ◆ “Move Destination Object” on page 159
- ◆ “Move Source Object” on page 160
- ◆ “Reformat Operation Attribute” on page 161
- ◆ “Remove Association” on page 162
- ◆ “Remove Destination Attribute Value” on page 163
- ◆ “Remove Source Attribute Value” on page 164
- ◆ “Rename Destination Object” on page 165 (when the selected object is DN or Association and Enter String)
- ◆ “Rename Source Object” on page 166 (when the selected object is DN or Association and Enter String)

- ◆ “Set Destination Attribute Value” on page 170 (when the selected object is DN or Association and Enter Value Type is not structured)
- ◆ “Set Destination Password” on page 171
- ◆ “Set Local Variable” on page 172
- ◆ “Set Operation Association” on page 173
- ◆ “Set Operation Class Name” on page 174
- ◆ “Set Operation Destination DN” on page 174
- ◆ “Set Operation Property” on page 175
- ◆ “Set Operation Source DN” on page 176
- ◆ “Set Operation Template DN” on page 176
- ◆ “Set Source Attribute Value” on page 177
- ◆ “Set Source Password” on page 178
- ◆ “Set XML Attribute” on page 180
- ◆ “Status” on page 181
- ◆ “Trace Message” on page 183

1 Create the argument using the nouns and verbs.


The noun and verbs can be combined to create the desired argument.


2 Click Finish.

Match Attribute Builder


The Match Attribute Builder enables you to select attributes and values used by the “Find Matching Object” on page 153 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:

- 1 Select the action of *find matching object*.
- 2 Select the scope of the search for the matching objects. Select from *entry*, *subordinates*, or *subtree*.
- 3 Specify the DN of the starting point for the search.
- 4 Click the *Edit match attributes* icon  to launch the Match Attribute Builder.

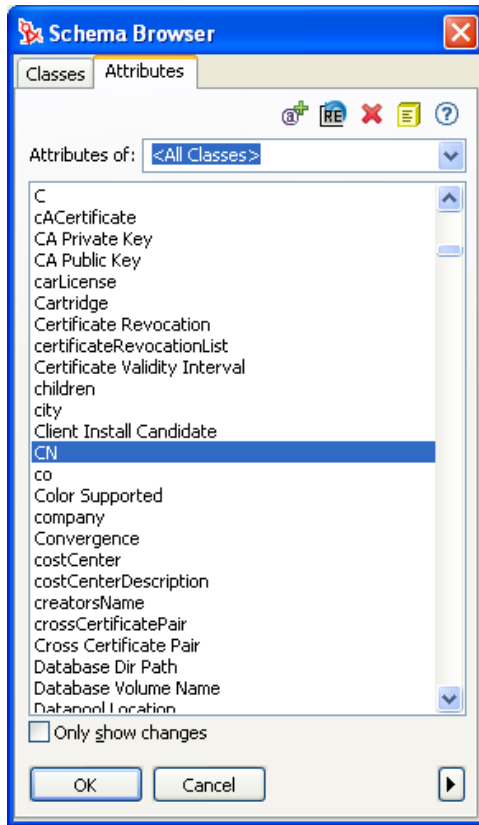
Do 

Select scope:

Enter DN: 


Enter match attributes: 

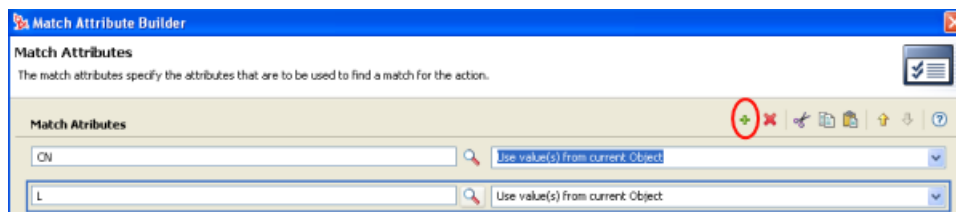
5 Click the *Browse attributes*  icon to launch the Schema Browser.



6 Click the *Attributes* tab, then browse to and select the desired attribute.


7 Click *OK*.

If you want to add more than one attribute, click the *Append new item* icon  to add another line.



8 Click *Finish*.

Action Argument Component Builder

To launch the Action Argument Component Builder, select one of the following actions when the *Enter value type* selection is *structured*, then click the *Edits components* icon .

- ♦ “Add Destination Attribute Value” on page 141
- ♦ “Add Source Attribute Value” on page 144
- ♦ “Reformat Operation Attribute” on page 161

- ◆ “Remove Destination Attribute Value” on page 163
- ◆ “Remove Source Attribute Value” on page 164
- ◆ “Set Destination Attribute Value” on page 170
- ◆ “Set Source Attribute Value” on page 177

Figure 2-6 Add Destination Attribute Value Action

Do ?

Enter attribute name: * 🔍

Enter class name: 🔍

Select mode: ▾

Select object: ▾

Enter DN: * 📄

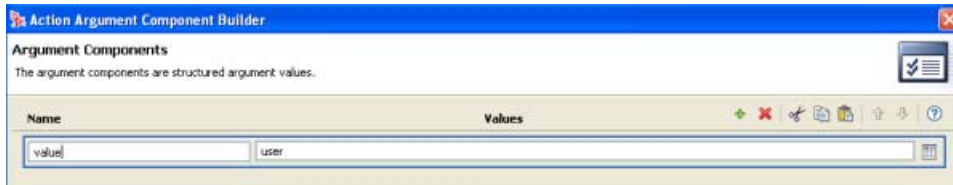
Enter value type: ▾

Enter components: * 📄

1 Click the *Edit the components* icon 📄 when the value type is set to structured.

2 Create the value of the action component.

You can enter in the value, or click on the *Edit the arguments* 📄 icon to create the value in the Argument Builder.



3 Click *Finish*.

Argument Value List Builder

To launch the Argument Value List Builder, select the following action, then click the *Edit the arguments* icon 📄.

- ◆ **Set Default Attribute Value**


Figure 2-7 Set Default Attribute Value

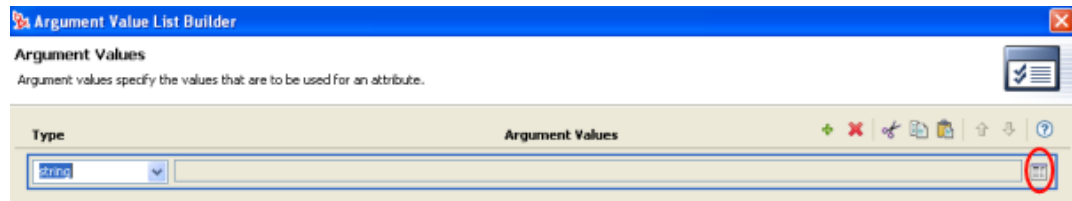
Do ?



Enter attribute name: * 🔍

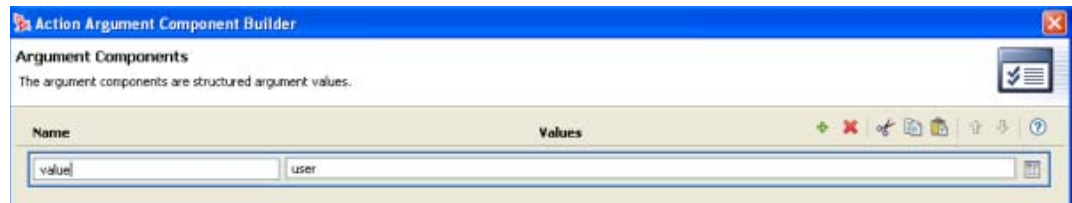
Write back: ▾

Enter argument values: * 📄

- 1 Select the type of the value: *counter, dn, int, interval, octet, state, string, structured, teleNumber, time*.
- 2 Click the *Edit the value lists* icon .




- 3 Click the *Edit the arguments* icon .
- 4 Create the value of the action component.
You can enter in the value, or click on the *Edit the arguments*  icon to create the value in the Argument Builder.




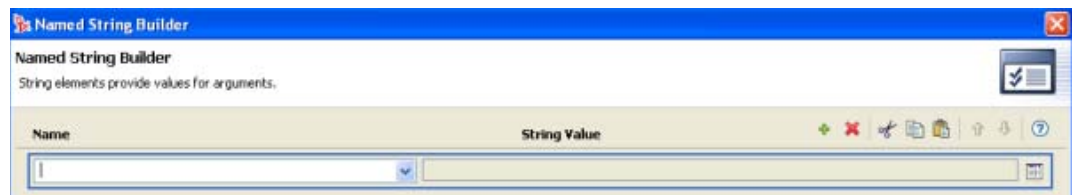
- 5 Click *Finish*.

Named String Builder

To launch the Named String Builder, select one of the following actions, then click the *Edit the strings* icon .

- ♦ **Generate Event**
- ♦ **Send Email**
- ♦ **Send Email From Template**

- 1 Select the name of the string from the drop-down list.
- 2 Create the value for the string by clicking the *Edit the arguments* icon  to launch the Argument Builder.



- 3 Click *Finish*.


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

Figure 2-8 E-mail Elements in the Send Mail Action





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


To launch the Condition Argument Component Builder, select one of the following conditions, then you must select the structured selection for Mode in order to see the *Launch ArgComponent Builder* icon .

- ◆ If Attribute
- ◆ If Destination Attribute
- ◆ If Source Attribute
- ◆ If Operation Attribute

Condition 

Name * 

Operator *

Mode 

Value

1 Specify the name and value of the condition component.

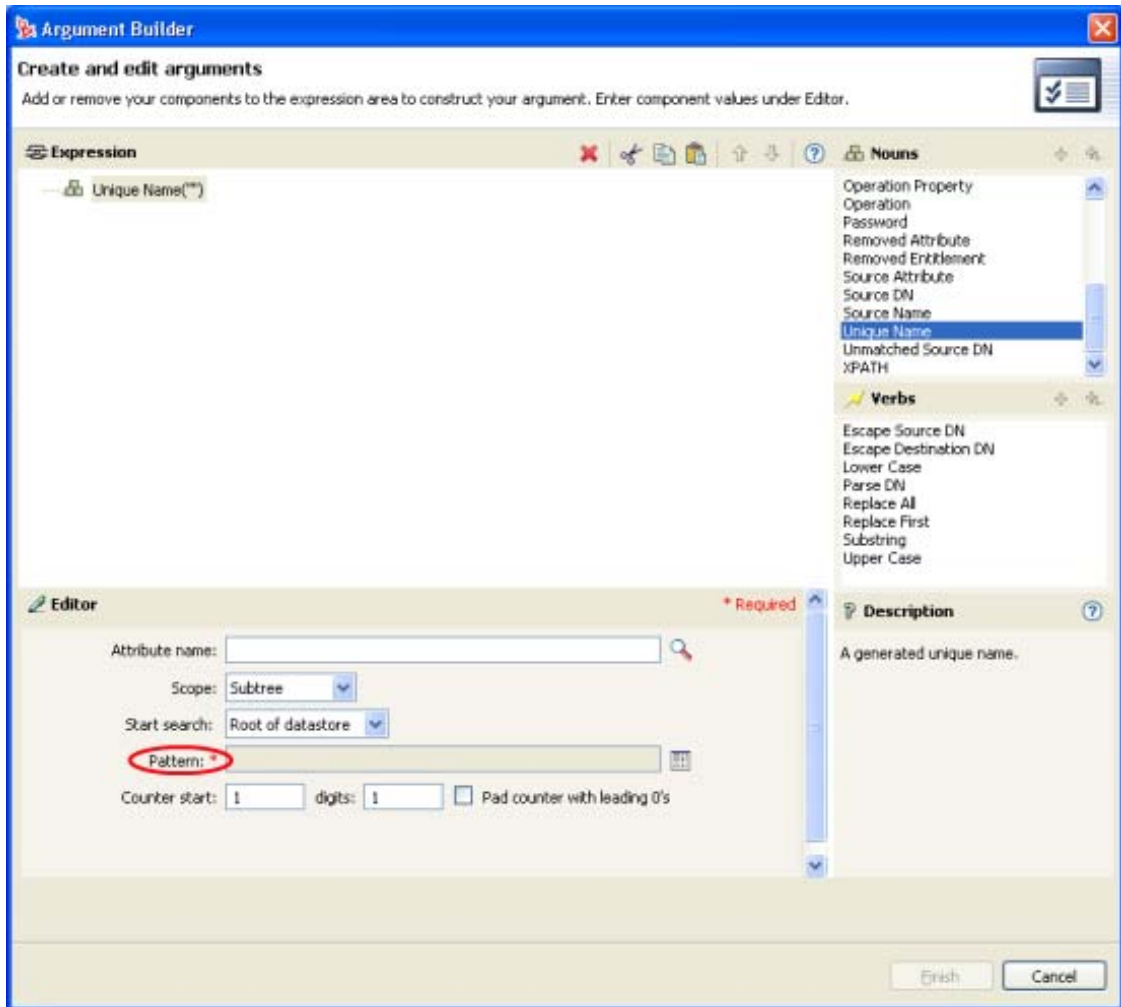




2 Click *Finish*.

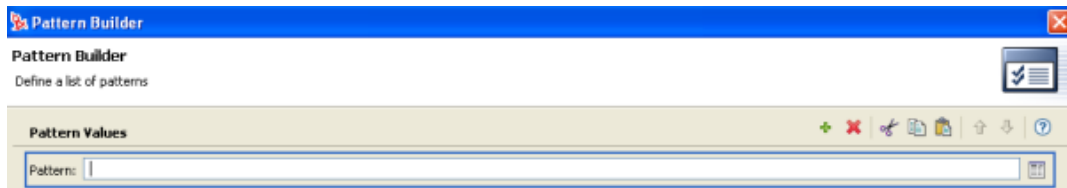
Pattern String Builder

You can 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 where you can click to launch the Pattern String Builder.

Figure 2-9 Unique Name Token in the Argument Builder



- 1 Click the *Edit patterns* icon  to launch the Pattern Builder.
- 2 Specify the pattern or click the *Edit the arguments* icon  to use the Argument Builder to create the pattern.

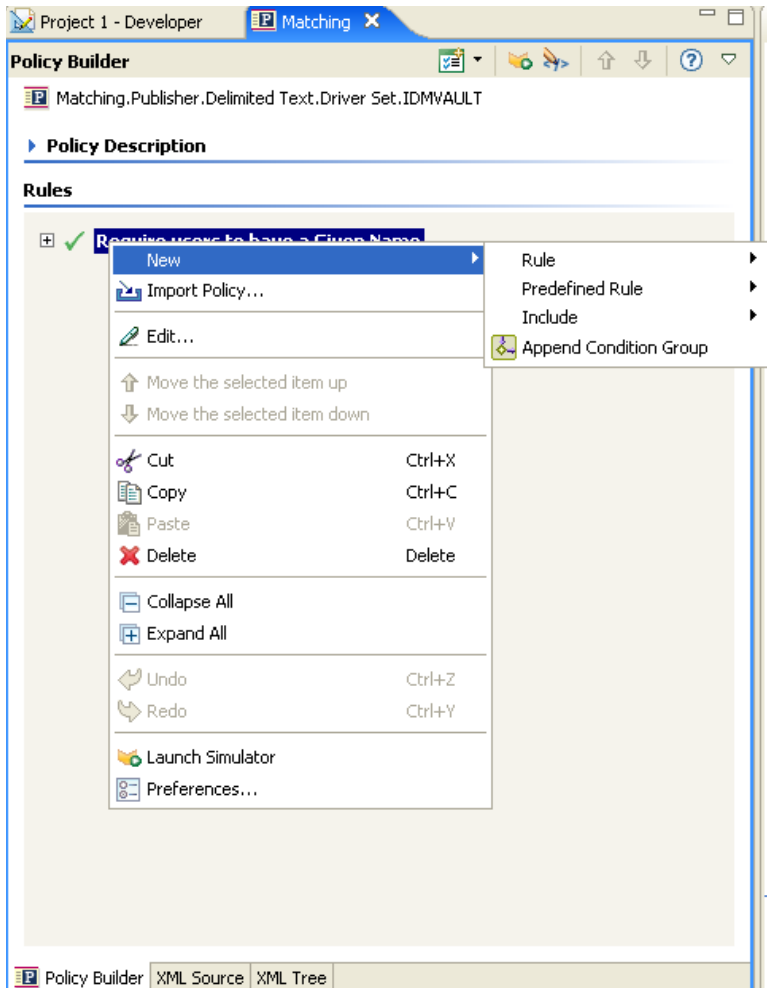


- 3 Click *Finish*.

2.2.5 Editing 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 2-10 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 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.
<i>Copy and drop</i>	Select the item, press Ctrl, then drag the item.

Operation	Description
<i>Cut</i>	Cuts the selected item and copies it to the Clipboard.
<i>Delete</i>	Deletes the selected item.
<i>Disable</i>	Disables a rule, condition, or action. Click the  icon.
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.
<i>Enable</i>	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.
<i>Move and drop</i>	Enables you to select and move an item. Select the item, then drag it.
<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 > Condition Group</i>	Creates a new condition group after a selected item.
<i>New > Include</i>	Creates a new Include after a selected item.
<i>New > Predefined Rule</i>	Inserts a predefined rule.
<i>New > 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.
<i>Select</i>	Click any item to select it.

KeyBoard Support

You can move through the Policy Builder with keystrokes as well as using the mouse. The supported keystrokes are listed below.

Table 2-4 Keyboard Support in the Policy Builder

Keystroke	Description
Ctrl+C	Copies the selected item into the Clipboard.

Keystroke	Description
Ctrl+X	Cuts the selected item and adds it to the Clipboard.
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.
Ctrl+Z	Undo
Ctrl+Y	Redo

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.

Policy Name:

Next Policy:

- 4 Click *OK*.

Saving Your Work

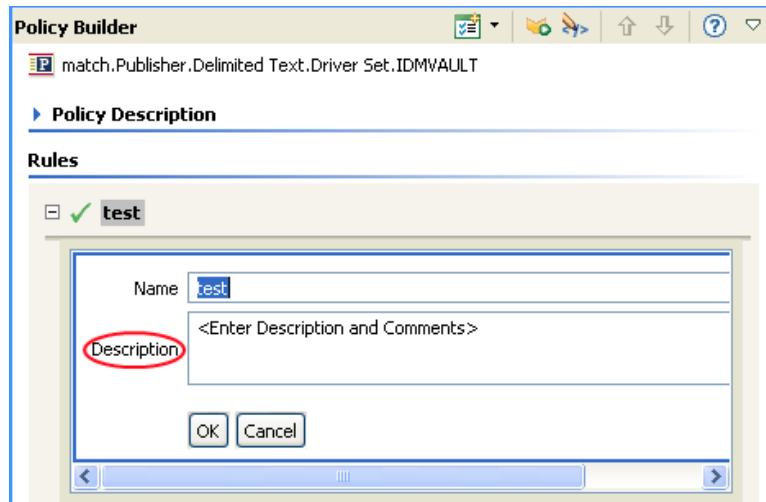
Do one of the following:

- ◆ 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.
- ◆ Press Ctrl+S.

Policy Description

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

Figure 2-11 Policy Description



2.2.6 Using Predefined Rules

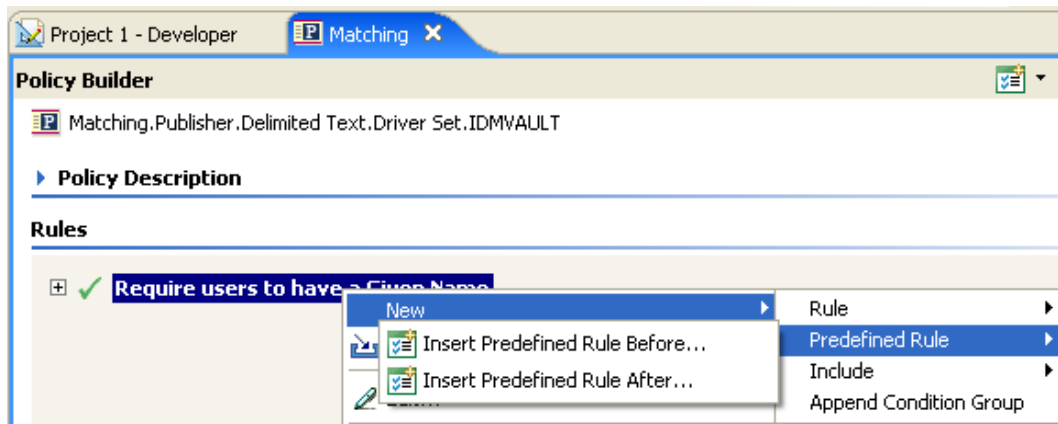
Designer includes twenty predefined rules. You can import and use these rules as well as create your own rules. These rules include common tasks that administrators use. You need to provide information specific to your environment to customize the rules.

- ◆ “Command Transformation - Create Departmental Container - Part 1 and Part 2” on page 76
- ◆ “Command Transformation - Publisher Delete to Disable” on page 78
- ◆ “Creation - Require Attributes” on page 79
- ◆ “Creation - Publisher - Use Template” on page 81
- ◆ “Creation - Set Default Attribute Value” on page 82
- ◆ “Creation - Set Default Password” on page 83
- ◆ “Event Transformation - Scope Filtering - Include Subtrees” on page 85
- ◆ “Event Transformation - Scope Filtering - Exclude Subtrees” on page 86
- ◆ “Input or Output Transformation - Reformat Telephone Number from (nnn) nnn-nnnn to nnn-nnn-nnnn” on page 88
- ◆ “Input or Output Transformation - Reformat Telephone Number from nnn-nnn-nnnn to (nnn) nnn-nnnn” on page 89
- ◆ “Matching - Publisher Mirrored” on page 90
- ◆ “Matching - Subscriber Mirrored - LDAP Format” on page 92
- ◆ “Matching - By Attribute Value” on page 94
- ◆ “Placement - Publisher Mirrored” on page 95
- ◆ “Placement - Subscriber Mirrored - LDAP Format” on page 97
- ◆ “Placement - Publisher Flat” on page 98

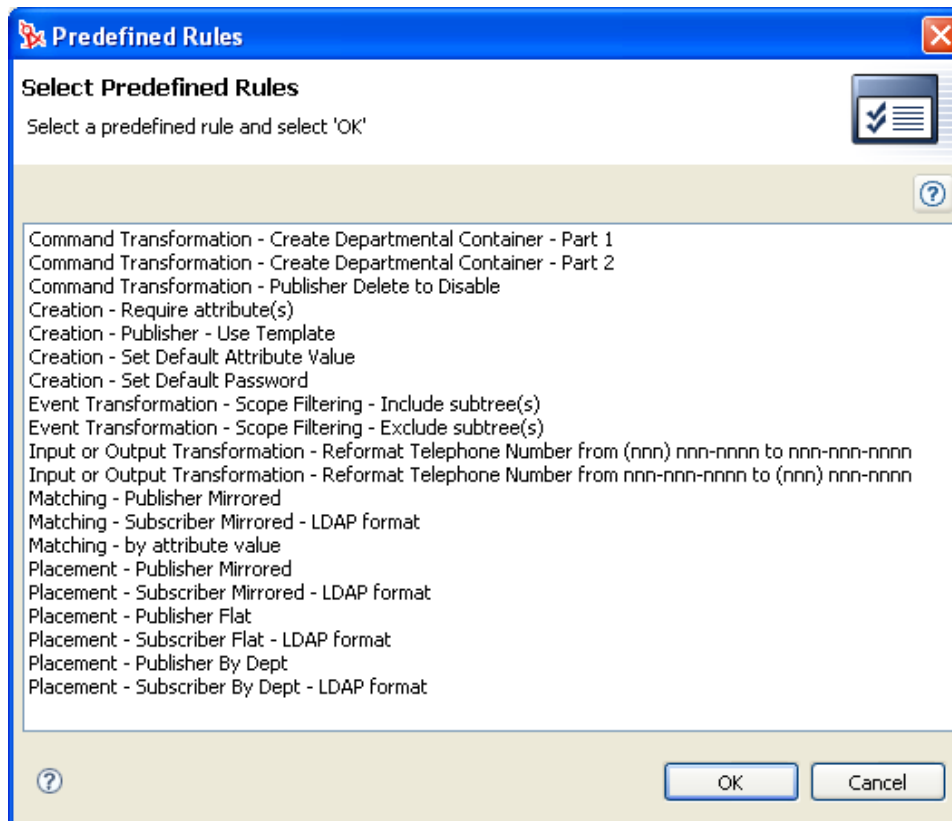
- ◆ “Placement - Subscriber Flat - LDAP Format” on page 100
- ◆ “Placement - Publisher By Dept” on page 101
- ◆ “Placement - Subscriber By Dept - LDAP Format” on page 103

To access the predefined rules:

- 1 In the Policy Builder, right-click and select *New > Predefined Rules > Insert Predefined Rule Before* or *Insert Predefined Rule After*.



The Predefined Rules dialog box displays a list of the available rules.




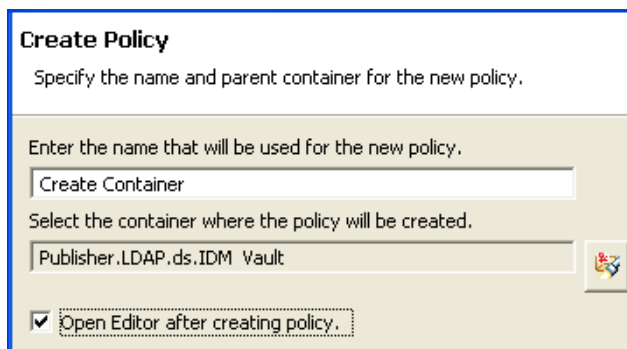
Command Transformation - Create Departmental Container - Part 1 and Part 2

Creates a department container in the destination data store, if one does not exist. Implement the rule on the Command Transformation policy in the driver. You can implement the rule on either the Subscriber or the Publisher channel or on both channels.

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 76](#).

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 view, then click *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.

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

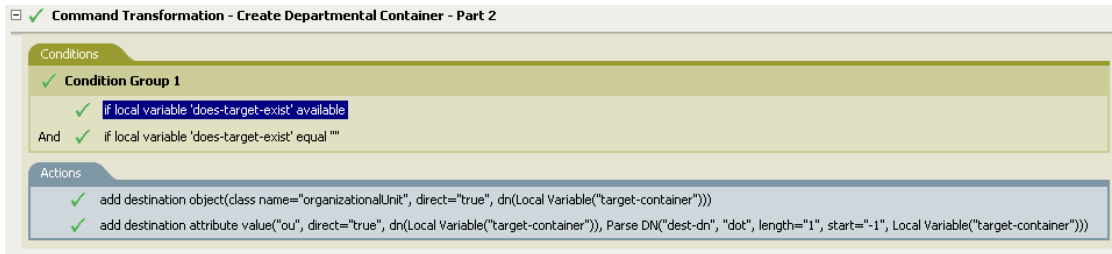
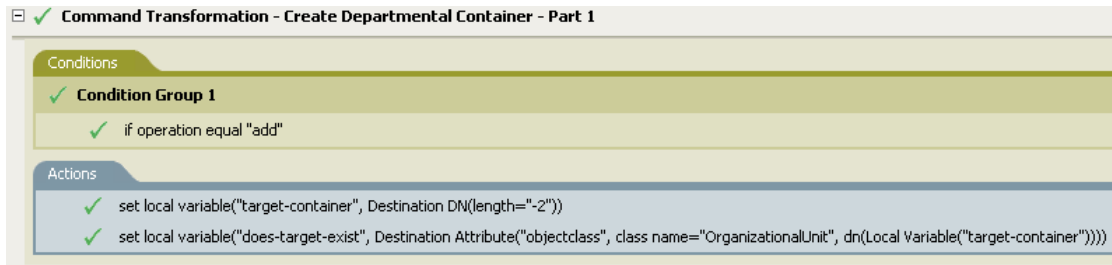
Open Editor after creating policy.

- 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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 2 Select *Command Transformation - Create Department Container - Part 1*, then click *OK*.
- 3 Right-click in Policy Builder and click *New > Predefined Rule > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 4 Select *Command Transformation - Create Department Container - Part 2*, then click *OK*.

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



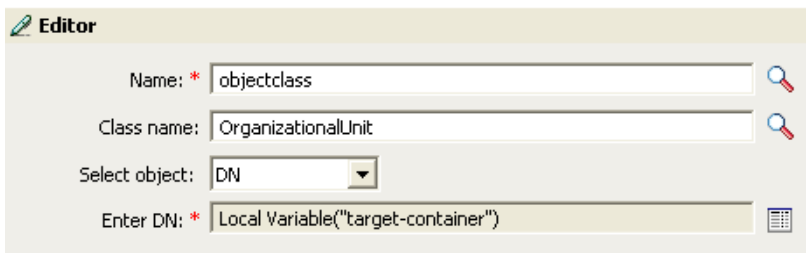
There is no information to change in the rules that are specific to your environment.

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

How 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 event. When the Add event 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.




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 local variable of target-container. It uses the source format as the destination DN and the destination format is dot format.

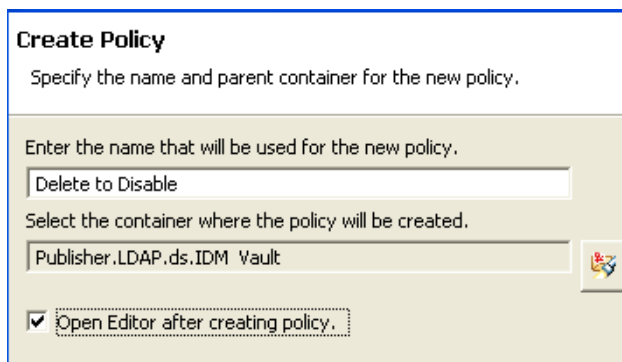
Command Transformation - Publisher Delete to Disable

Transforms the Delete event for a user object into disabling the user object. Implement the rule on the Command Transformation policy in the driver. The rule needs to be implemented on the Publisher channel.

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 78](#).


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 view, then click *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.

Enter the name that will be used for the new policy.
Delete to Disable

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

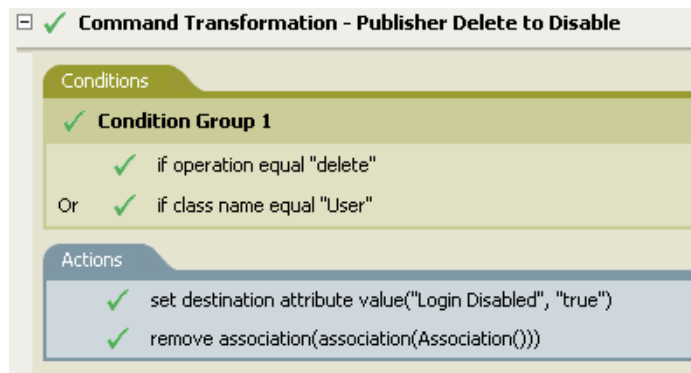
Open Editor after creating policy.

- 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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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.

How the Rule Works


The rule is used when a Delete event occurs in the connected data store. Instead of the user object being deleted in the Identity Vault, the User object is disabled. Anytime a Delete event occurs for a User object, the destination attribute value of Login Disabled is set to True and the association is removed from the User object. The User object can no longer log in into the Novell eDirectory tree, but the User object was not deleted.

Creation - Require Attributes

The rule does not allow user objects to be created unless the required attributes are populated. Implement the rule on the Creation policy in the driver. You can implement the rule on either the Subscriber or the Publisher channel or on both channels.

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 80](#).

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 view, then click *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.

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 and click *New > Predefined Rule > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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*.

Creation - Require attribute(s)

Conditions

Condition Group 1

if class name equal "User"

Actions

veto if operation attribute not available("[Enter name of required attribute]")

How the Rule Works

The rule is used when your business processes require a user to have specific attributes populated when the user object is created. When a user object is created, the rule vetoes the creation of the object unless the required attributes are provided. You can have one or more required attributes.


If you want more than one required attribute, right-click the action and select *New > Append Action*. Select *veto if operation attribute not available*, then browse to the attribute you want to require.

Creation - Publisher - Use Template

Allows 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. You can implement the rule only on the Publisher channel.

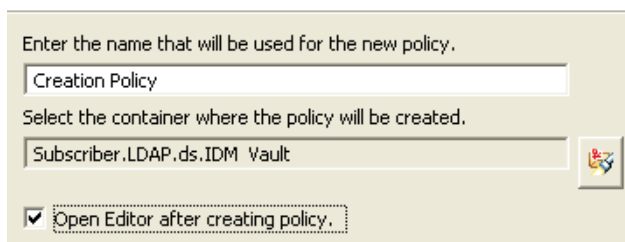
There are two steps involved in using the predefined rules: creating a policy in the Creation policy set and importing the predefined rule. If you already have a Creation policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 81](#).

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 view, then click *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.

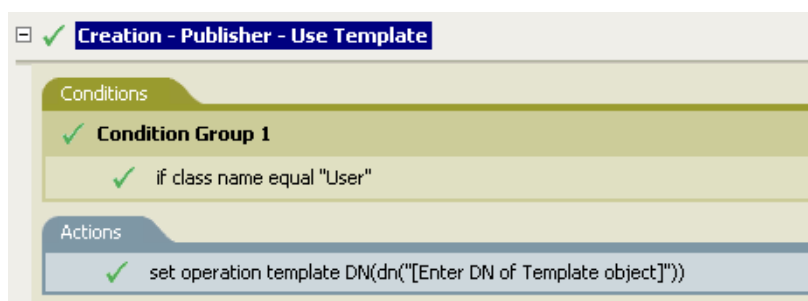


- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “Before editing this item you need to save. Do you wish to save the editor’s changes and continue?” Click *Yes*. 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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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 Rule Works

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


During the creation of User objects, the rule performs the action of the set operation template DN. The action calls the template object and creates the User object with the information in the template.

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 83](#).

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 view, 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.



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

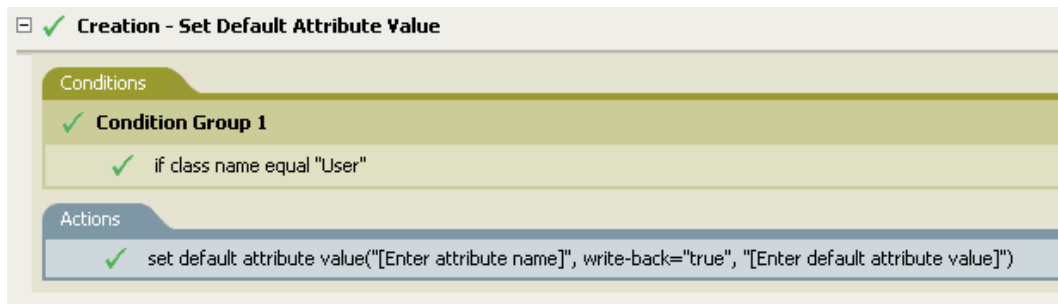
Select the container where the policy will be created.

 
 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 Creation policy is saved.

Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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 create.
- 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 for the attribute in the Argument Builder, then click *OK*.
- 11 Click *OK*.
- 12 Save the rule by clicking *File > Save*.



How the Rule Works

The rule is used when you want to create a User object with default attributes and values. When a User object is created, the rule sets the attribute and the value for that attribute.


If you want more than one attribute value defined, right-click the action and click *New > Append Action*. Select the action, set the default attribute value, and follow [Step 1 on page 83](#) through [Step 12 on page 83](#) 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 Creation policy in the driver. You can implement the rule on either the Subscriber or the Publisher channel or on both channels.

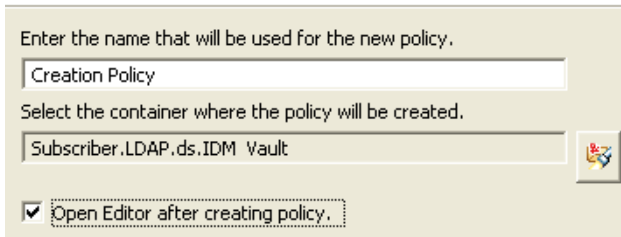
There are two steps involved in using the predefined rules: creating a policy in the Creation policy set and importing the predefined rule. If you already have a Creation policy that you want to add this rule to, skip to [“Importing the Predefined Rule” on page 84](#).

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 view, then click *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.



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

Creation Policy

Select the container where the policy will be created.

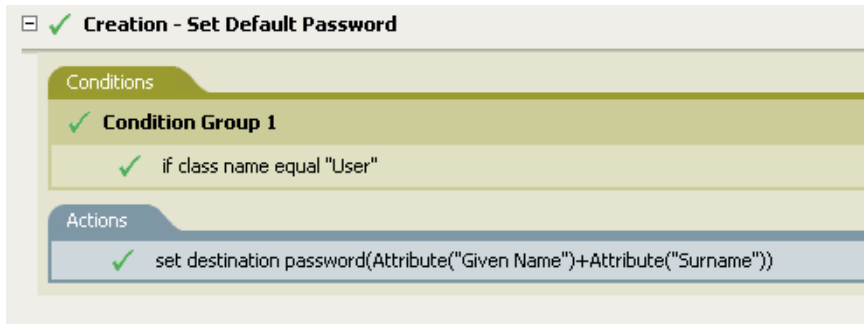
Subscriber.LDAP.ds.IDM Vault

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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 2 Select *Creation - Set Default Password*, then click *OK*.
- 3 Save the rule by clicking *File > Save*.



Creation - Set Default Password

Conditions

✓ Condition Group 1

✓ if class name equal "User"

Actions

✓ set destination password(Attribute("Given Name")+Attribute("Surname"))

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

How 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 except for the specific subtree. Implement the rule on the Event Transformation policy in the driver. You can implement the rule on either the Subscriber or the Publisher channel or on both channels.

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 \(page 86\)](#).

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 view, then click *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.

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

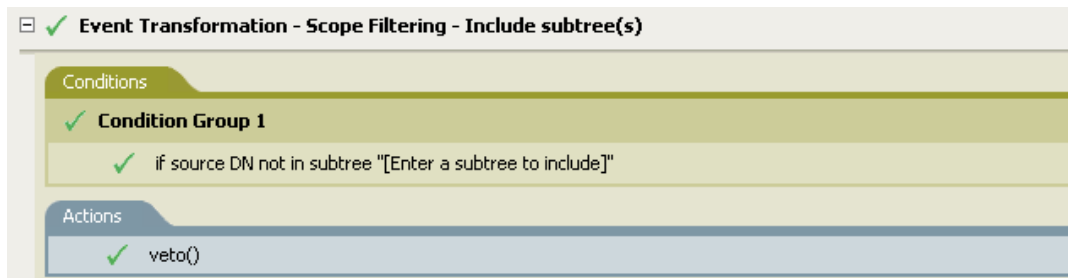
Select the container where the policy will be created.
Publisher.LDAP.ds.IDM Vault

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 Event Transformation policy is saved.

Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then select *New > Predefined Rule > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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 clicking *File > Save*.



How the Rule Works

The rule is used when you want to exclude part of the Identity Vault from synchronizing. It allows you to synchronize some objects and not other objects, without using the Filter. When an event occurs anywhere but in that specific part of the Identity Vault, it is vetoed.

Event Transformation - Scope Filtering - Exclude Subtrees

Excludes all events that occur in a specific subtree. Implement the rule on the Event Transformation policy in the driver. You can implement the rule on either the Subscriber or the Publisher channel or on both channels.

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 87](#).

Creating a Policy

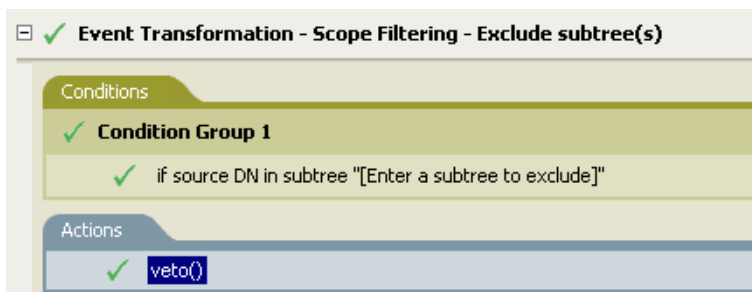
- 1 From the *Outline* view or the *Policy Flow* view, select the Publisher or Subscriber channel.
- 2 Select the Event Transformation policy set in Policy Set view, then click *Create or add a new policy to the Policy Set* icon **+** to create a new policy.
- 3 Click *Create a new policy*, then click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

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

- 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 where you want to exclude events from synchronizing, then click *OK*.
- 6 Click *OK*.
- 7 Save the rule by clicking *File > Save*.



How the Rule Works


The rule is used when you want to exclude part of the Identity Vault from synchronizing. It allows you to synchronize some objects and not other objects, without using the Filter. Anytime an event occurs in that specific part of the Identity Vault, it is vetoed.

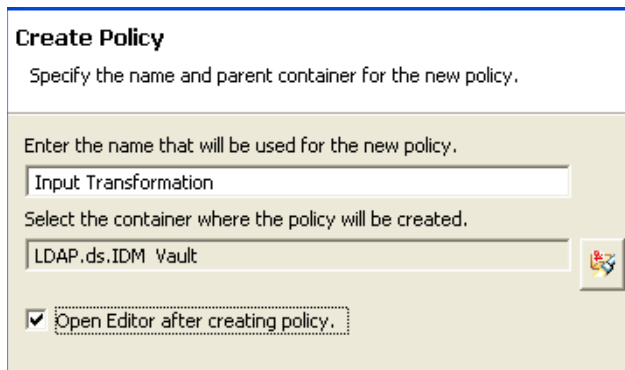
Input or Output Transformation - Reformat Telephone Number from (nnn) nnn-nnnn to nnn-xxx-xxxx

Transforms the format of the telephone number when a desired condition is met. Implement the rule on the Input or Output Transformation policy in the driver. You can implement the rule on either the Subscriber or the Publisher channel or on both channels.

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 88](#).

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 view, then click *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.

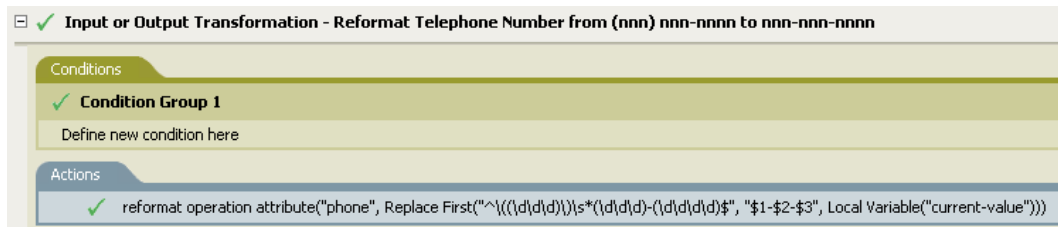


- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “Before editing this item you need to save. Do you wish to save the editor’s changes and continue?” Click *Yes*. 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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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 Rule Works


The rule is used when you want to reformat the telephone number. You define the condition that is to be met when the telephone number is reformatted.

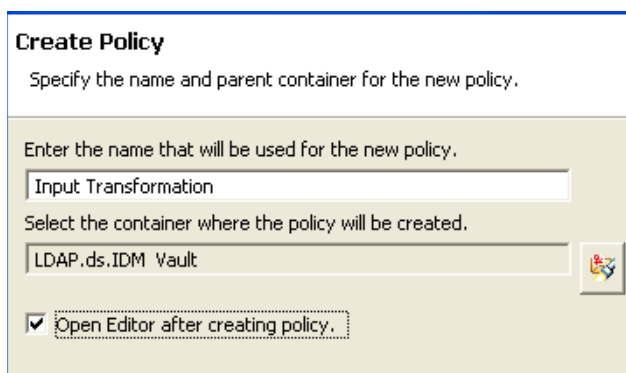
Input or Output Transformation - Reformat Telephone Number from nnn-xxx-xxxx to (nnn) nnn-nnnn

Transforms the format of the telephone number when a desired condition is met. Implement the rule on the Input or Output Transformation policy set. You can implement the rule on either the Subscriber or the Publisher channel or on both channels.

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 90](#).

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 view, then click *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.

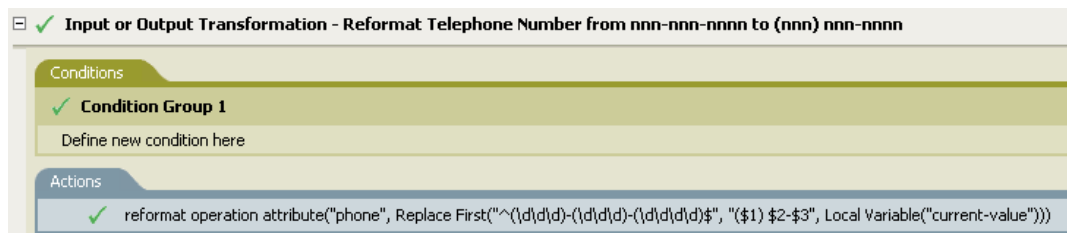


- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.

- 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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 2 Click *Input or Output Transformation - Reformat Telephone Number from nnn-xxx-nnnn to (nnn) nnn-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 Rule Works


The rule is used when you want to reformat the telephone number. You define the condition that is to be met when the telephone number is reformatted.

Matching - Publisher Mirrored

Matches for objects in the Identity Vault by using the mirrored structure in the data store from a specified point. Implement the rule on the Matching policy in the driver. You can implement the rule only on the Publisher channel.

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 \(page 91\)](#).

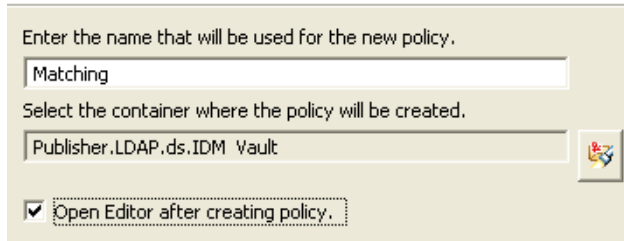
Creating a Policy

- 1 From the *Outline* view or the *Policy Flow* view, select the Publisher channel.
- 2 Select the Matching policy set in the Policy Set view, then click *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.

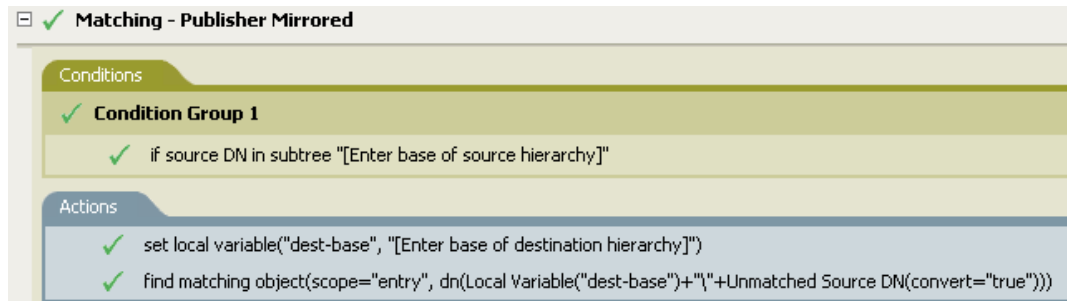


- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “Before editing this item you need to save. Do you wish to save the editor’s changes and continue?” Click *Yes*.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

Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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 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*.



How the Rule Works


Matches for objects in the Identity Vault by using the mirrored structure in the data store from a specified point. When an Add event occurs and the driver checks to see if the object exists, it starts checking at the specific DN in the data store. The driver then sets a local variable of dest-base to be the starting point in the Identity Vault that the structure is mirrored to in the data store. The driver then creates the context it is searching by adding the local variable of dest-base plus a \ and the source DN of the object. It creates the path it is looking for in the slash format.

Matching - Subscriber Mirrored - LDAP Format

Matches for objects in the data store by using the mirrored structure in the Identity Vault from a specified point. Implement the rule on the Matching 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 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 93](#).

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 view, then click *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

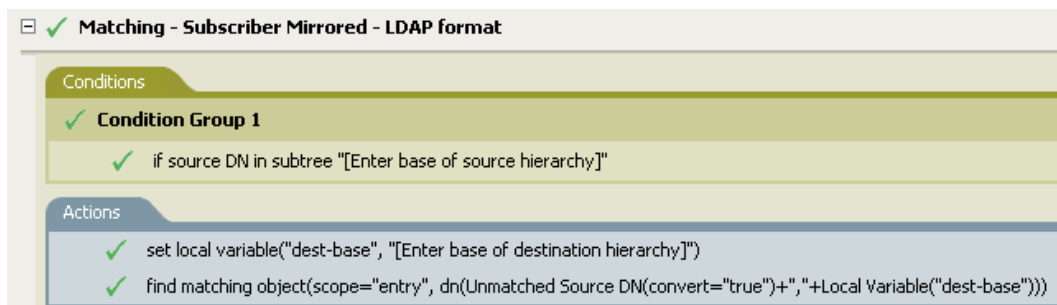
Specify the name and parent container for the new policy.

The screenshot shows a "Create Policy" dialog box. It has a title bar and a main area with the following elements: a text input field with the value "Matching" and a label "Enter the name that will be used for the new policy."; a dropdown menu with the value "Publisher.LDAP.ds.IDM vault" and a label "Select the container where the policy will be created."; a checkbox labeled "Open Editor after creating policy." which is checked.

- 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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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 Rule Works


Matches for objects in the data store by using the mirrored structure in the Identity Vault from a specified point. When an Add event occurs and the driver checks to see if the object exists, it starts checking at the specific DN in the Identity Vault. The driver then sets a local variable of *dest-base* to be the starting point in the data store that the structure is mirrored to in the Identity Vault. The driver then creates the context it is searching by adding the source DN of the object plus a, and the local variable of *dest-base*. It creates the path it is looking for in LDAP format.

Matching - By Attribute Value

Matches for objects by specific attribute values. Implement the rule on the Matching policy in the driver. You can implement the rule on either the Subscriber or the Publisher channel or on both channels.

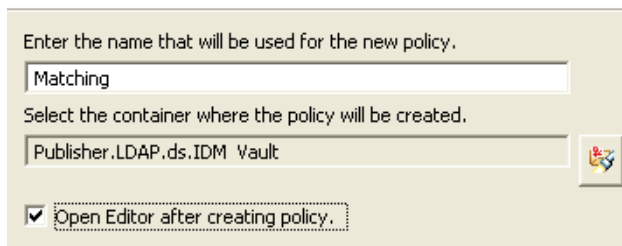
There are two steps involved in using the predefined rules; creating a policy in the Matching policy set and importing the predefined rule. If you already have a Matching policy that you would like to add this rule to, skip to [“Importing the Predefined Rule” on page 94](#).

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 view, then click *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.

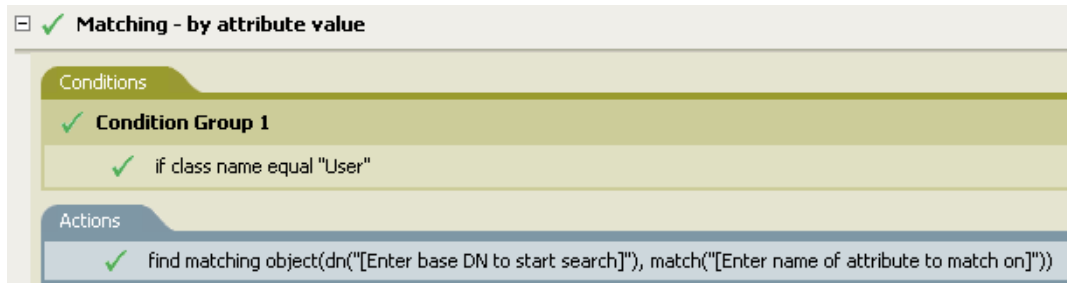


- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “Before editing this item you need to save. Do you wish to save the editor’s changes and continue?” Click *Yes*.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

Importing the Predefined Rule

- 1 Right-click in the Policy Builder, then click *New > Predefined Rule > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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*.



How the Rule Works


Matches for User objects by attributes. When a User object is synchronized, the driver uses the rule to check and see if the specified attributes exist. If they attributes do not exist, a new User object is created.

Placement - Publisher Mirrored

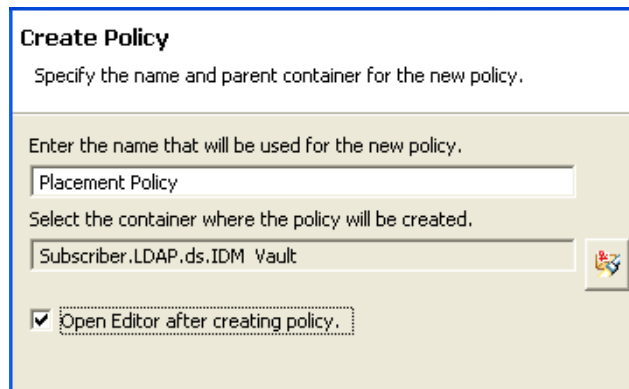
Places objects in the Identity Vault by using the mirrored structure in the data store from a specified point. Implement the rule on the Placement policy in the driver. You can implement the rule only on the Publisher channel.

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 96](#).

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* 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.


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

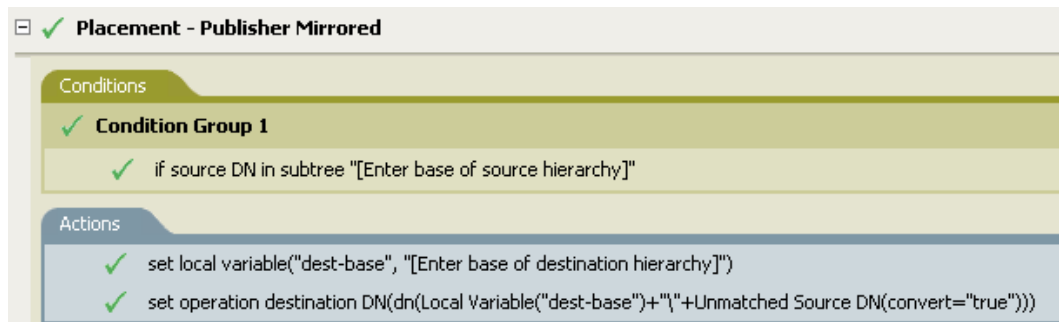
Open Editor after creating policy.

- 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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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 Rule Works


If the User object resides in the source hierarchy, the object is placed in the mirrored structure from the data store. The placement starts at the point that the local variable `dest-base` is defined. It places the User object in the location of `dest-base\unmatched source DN`. The rule uses the slash format.

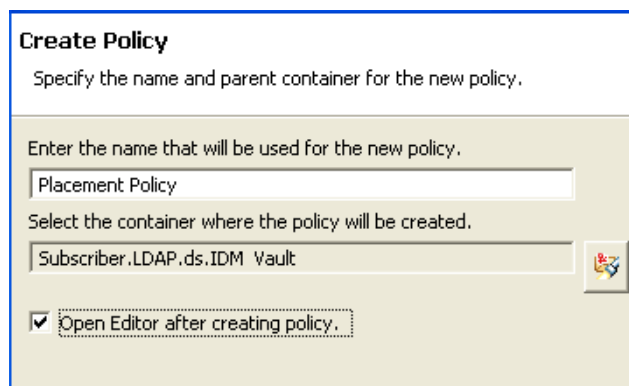
Placement - Subscriber Mirrored - LDAP Format

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 98](#).

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 view, then click *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.

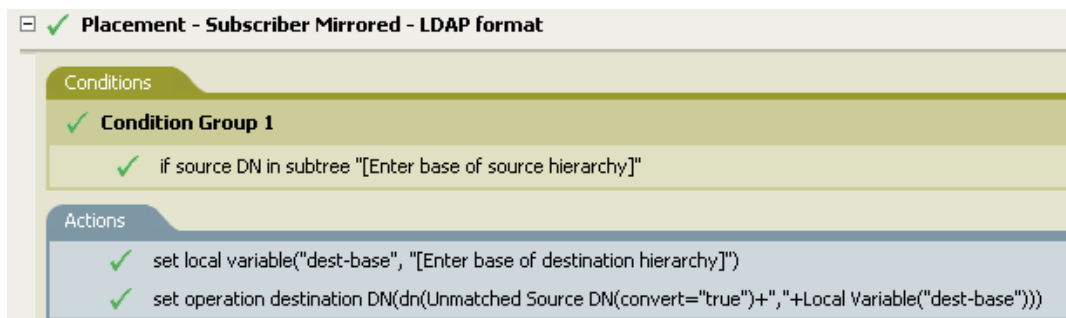


- 6 Select *Open Editor after creating policy*, then click *Next*.

- 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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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*.



How the Rule Works


If the User object resides in the source hierarchy, then the object is placed in the mirrored structure from the Identity Vault. The placement starts at the point that the local variable *dest-base* is defined. It places the User object in the location of unmatched source DN, *dest-base*. The rule uses LDAP format.

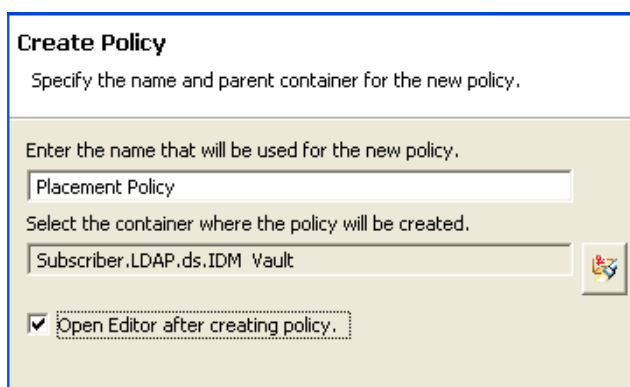
Placement - Publisher Flat

Places objects from the data store into one container in the Identity Vault. Implement the rule on the Placement policy in the driver. You can implement the rule only on the Publisher 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 99](#).


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 view, then click *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.

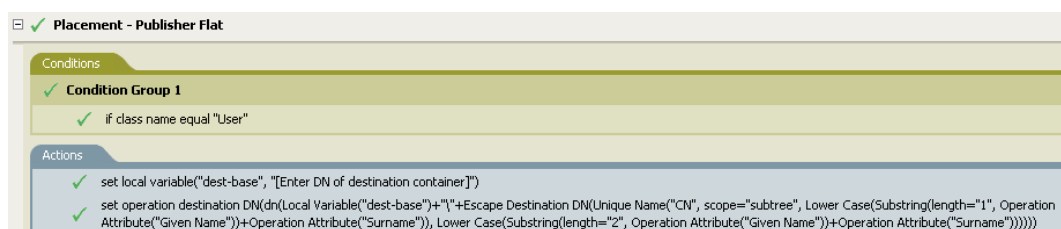


- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “Before editing this item you need to save. Do you wish to save the editor’s changes and continue?” Click *Yes*. 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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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 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 the `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.

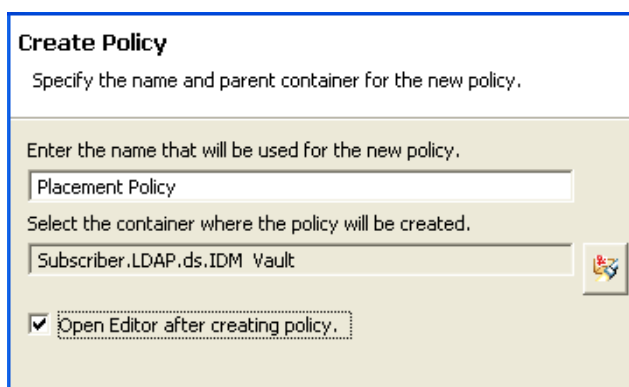
Placement - Subscriber Flat - LDAP Format

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 101](#).

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 view, then click *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.

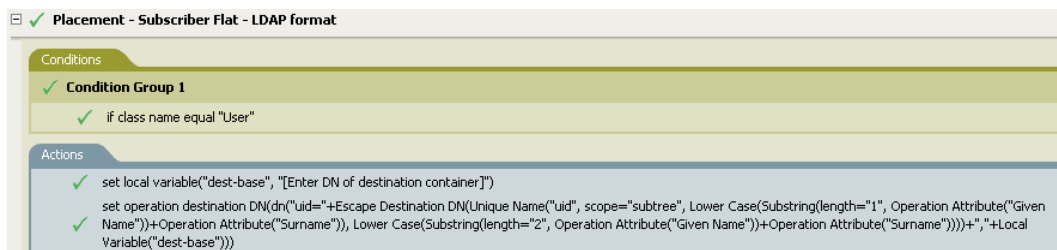


- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.

- 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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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 where 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*.



How 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 in lowercase. The rule uses LDAP format.


Placement - Publisher By Dept

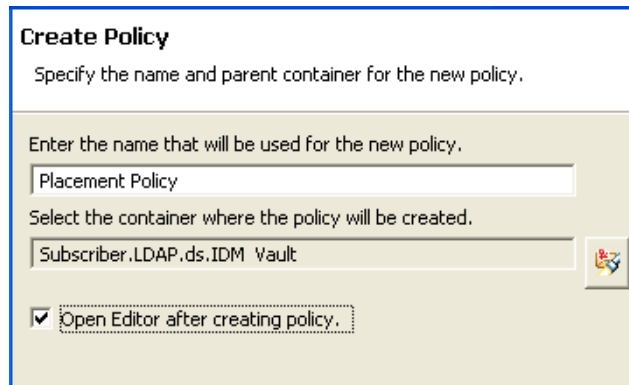
Places objects from one container in the data store into multiple containers in the Identity Vault. Implement the rule on the Placement policy in the driver. You can implement the rule only on the Publisher channel.

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 102](#).

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 view, then click *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.

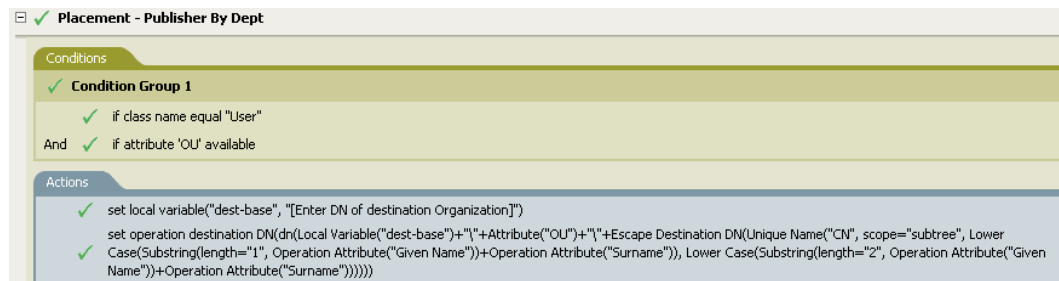


- 6 Select *Open Editor after creating policy*, then click *Next*.
- 7 Select *DirXML Script* for the type of policy, then click *Finish*.
- 8 A file conflict window appears with the message “Before editing this item you need to save. Do you wish to save the editor’s changes and continue?” Click *Yes*. 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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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 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 child containers must be associated for the user objects to be placed. 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 in 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 based 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 104](#).


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 view, then click *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.

- 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 > Insert Predefined Rule Before* or *Insert Predefined Rule After*.
- 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 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 child containers must be associated for the User objects to be placed. 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

The Policy Simulator allows you to execute a policy at any point in the flow of the driver and see the results without implementing the policy in the Identity Vault. You can test the policies without affecting the production environment or the connected system.

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

- ♦ “Accessing the Policy Simulator” on page 105
- ♦ “Using the Policy Simulator” on page 107

The Policy Simulator uses XML. The eDirectory document type definition file (`nds.dtd`) defines the schema of the XML documents that the Metadirectory engine can process. XML documents that do not conform to this schema generate errors. To verify whether the document conforms to the `nds.dtd` and find information about why errors are occurring, 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).

The Policy Simulator cannot simulate the initial policy sets from application drivers such as SOAP and Delimited text. These drivers use comma-separated files or text files as input, and the XML or XDS is derived from policies in the policy chain. Currently, the Policy Simulator only accepts valid XML or XDS as input. Additional functionality is being considered for future releases.

Accessing the Policy Simulator

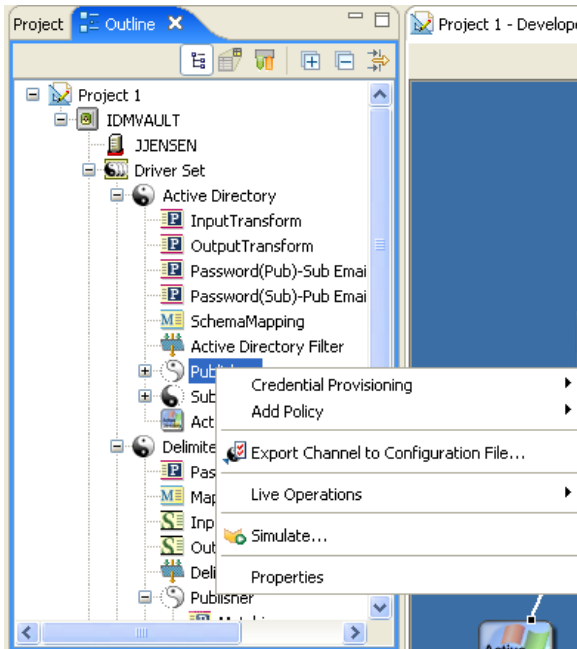
The Policy Simulator can be accessed in three different ways:

- ♦ “Outline View” on page 105
- ♦ “Policy Flow” on page 106
- ♦ “Editors” on page 106


Outline View

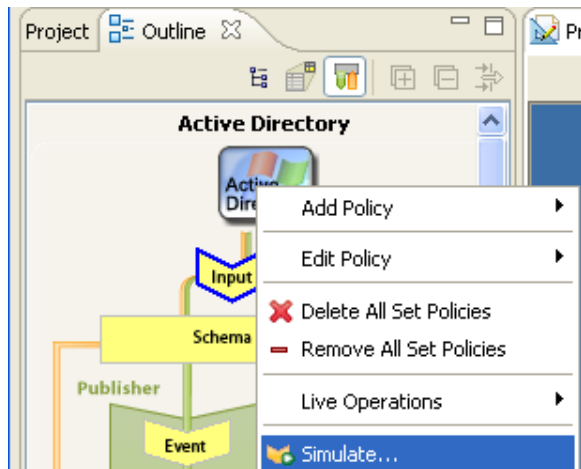
- 1 Click the *Show Model Outline* icon .

- 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 *Show 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.

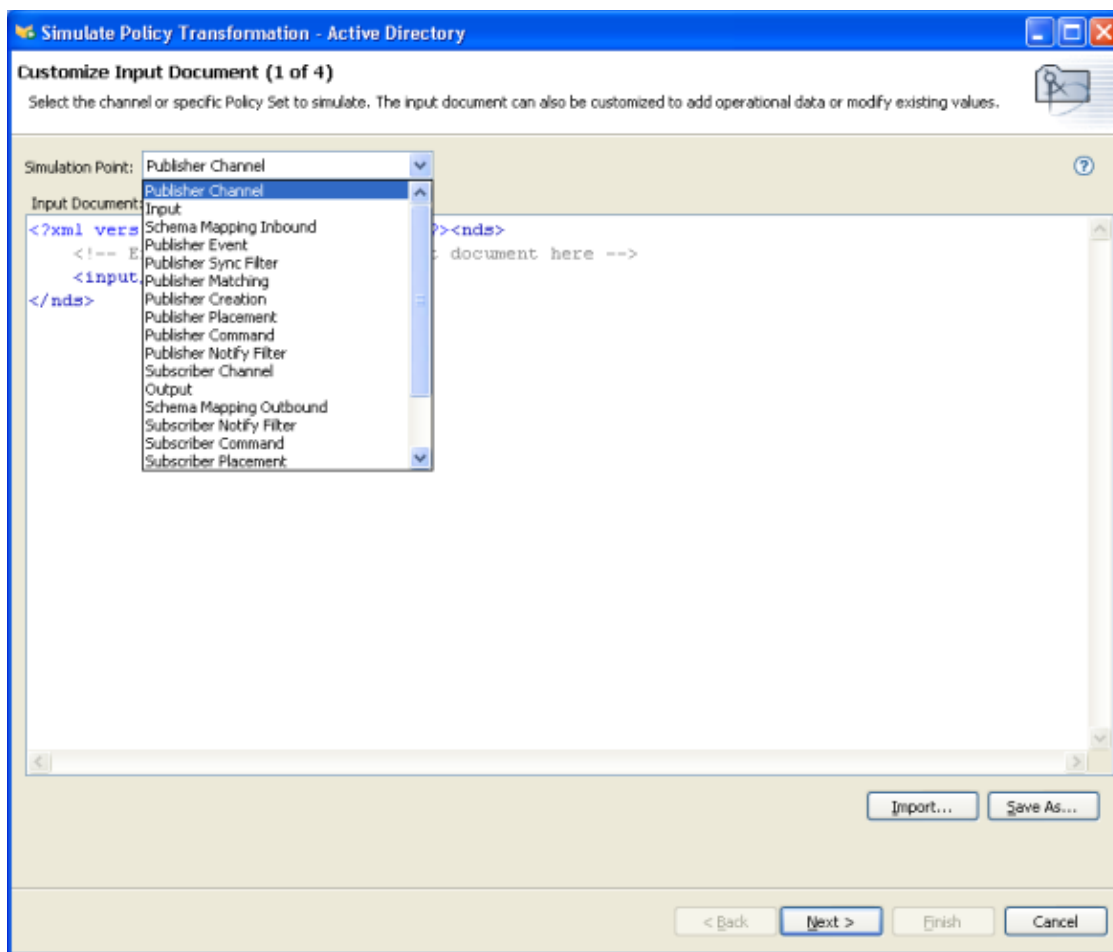
Using the Policy Simulator

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.

To use the Policy Simulator:

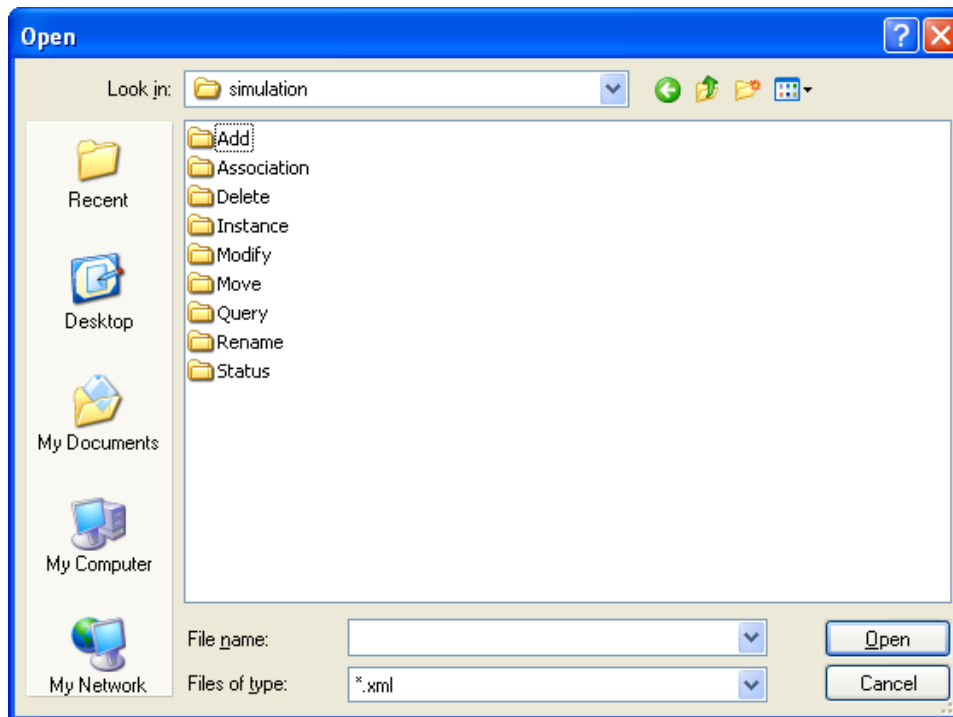
- 1 From the Simulation Point drop-down list, select the place in the driver flow that you want to test the policy. You can select any of the following items: Publisher Channel, Subscriber Channel, Input, Schema Mapping, Event, Sync Filter, Matching, Creation, Placement, Command and Notify Filter.

If you select a specific policy or rule to test, the Simulation Point option only shows *To NDS* or *From NDS*.



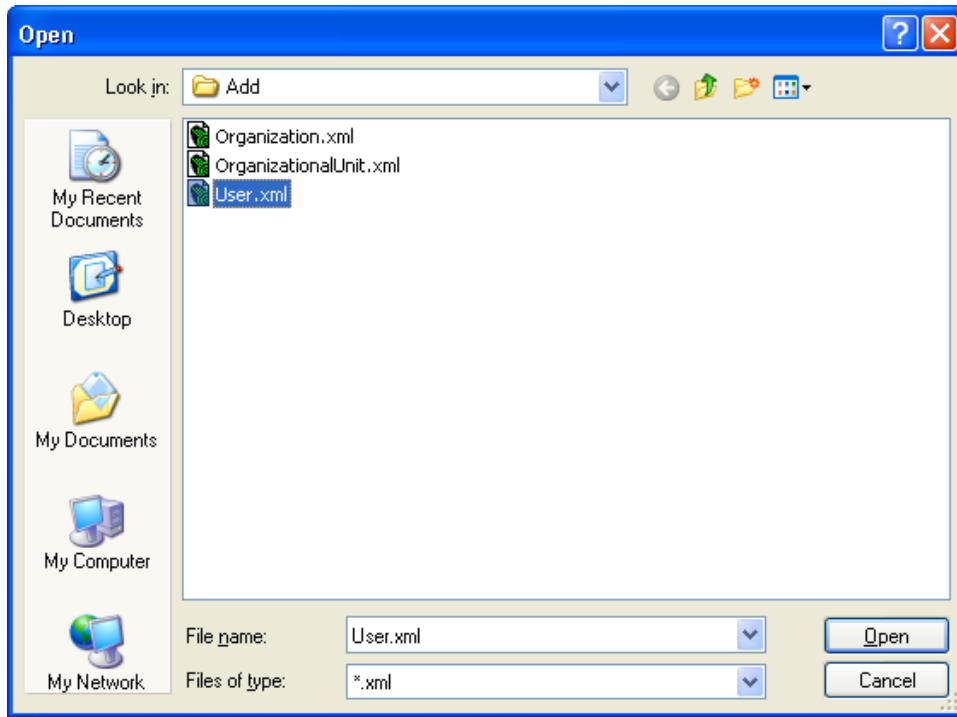
- 2 Select *Import*, then browse to and select a file to test.

Designer comes with sample event files you can use. The files are located in the plug-in `com.novell.designer.idm.policy\simulation`. The event are Add, Association, Delete Instance, Modify, Move, Query, Rename and Status.



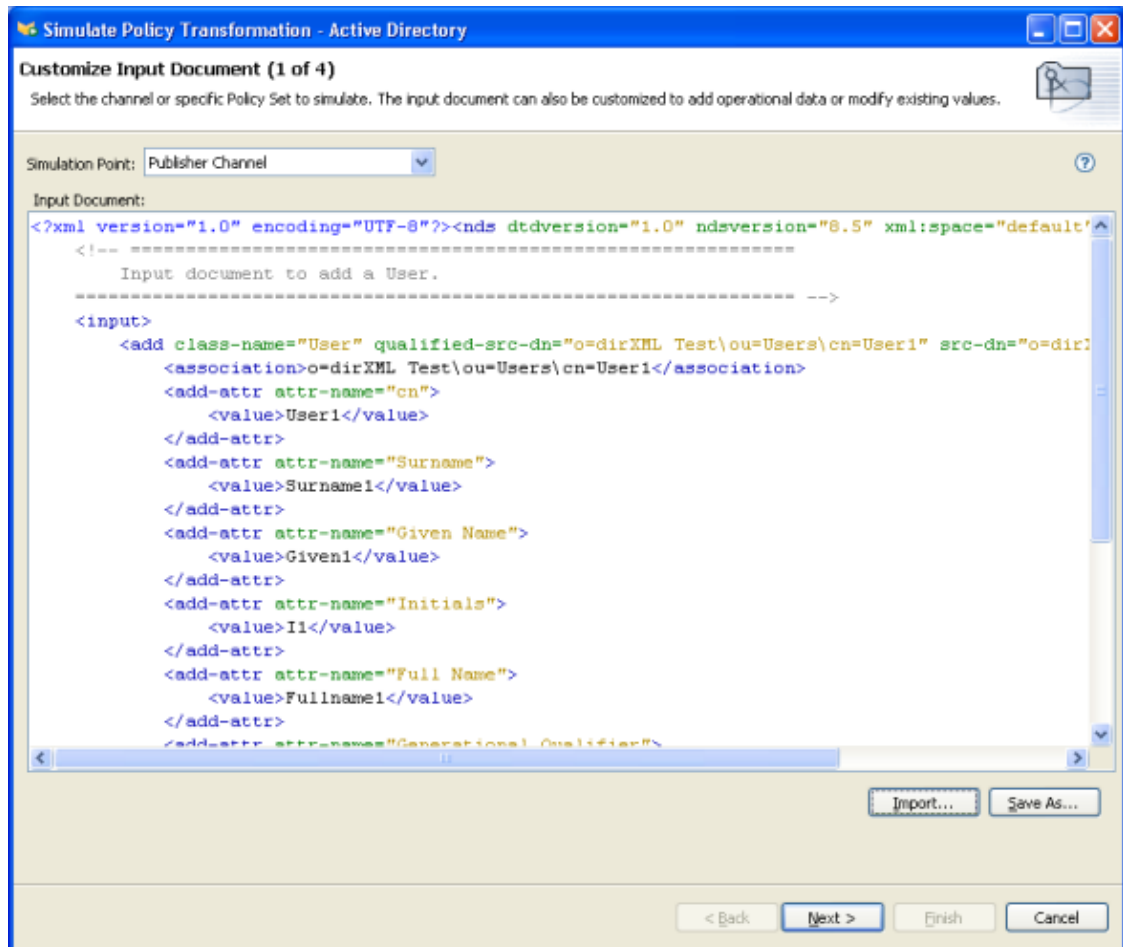
- 3 Double-click a folder and to display the available events. Each event has different files you can select. For example, if you select *Add*, you have three options: `Organization.xml`,

OrganizationalUnit.xml, and User.xml. The file indicates the event. If you select User.xml, it is an Add event for a user object.

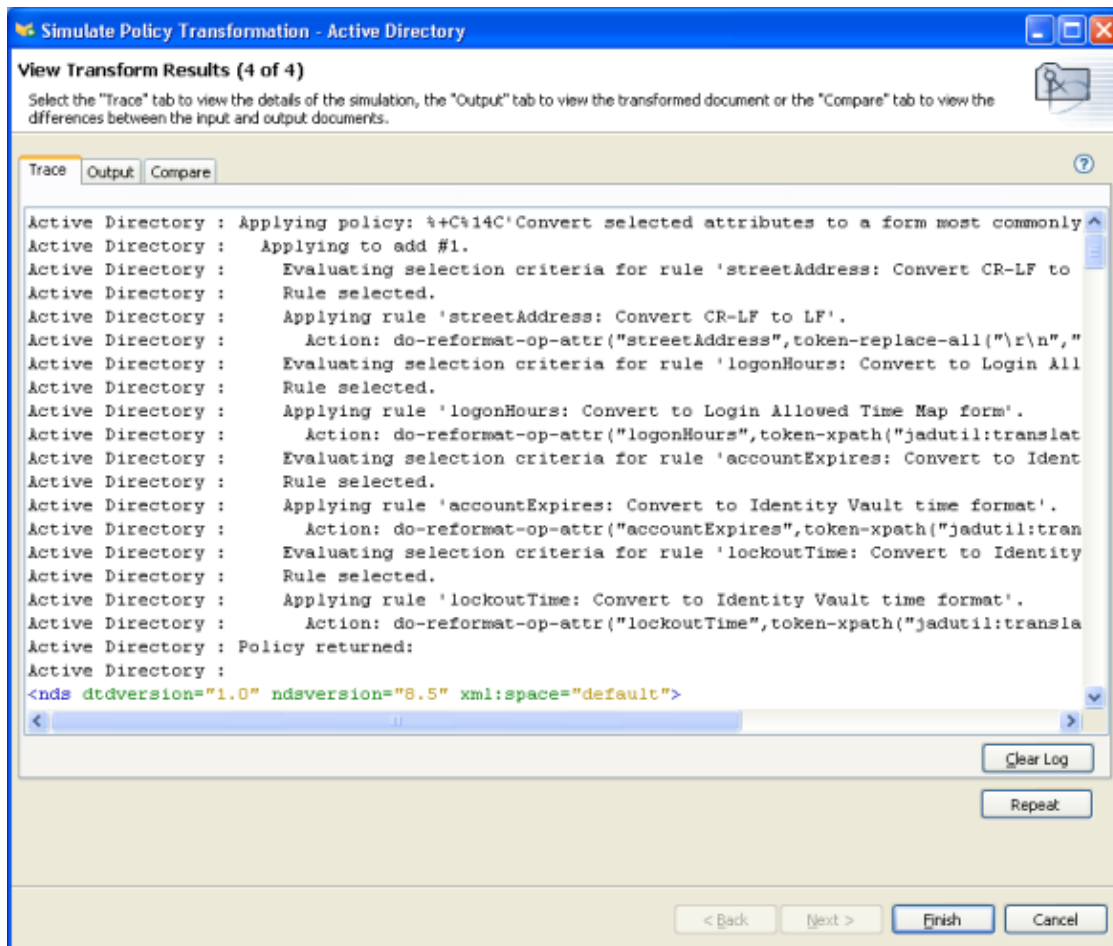


4 Select a file, then click *Open* to display the input document in the window.

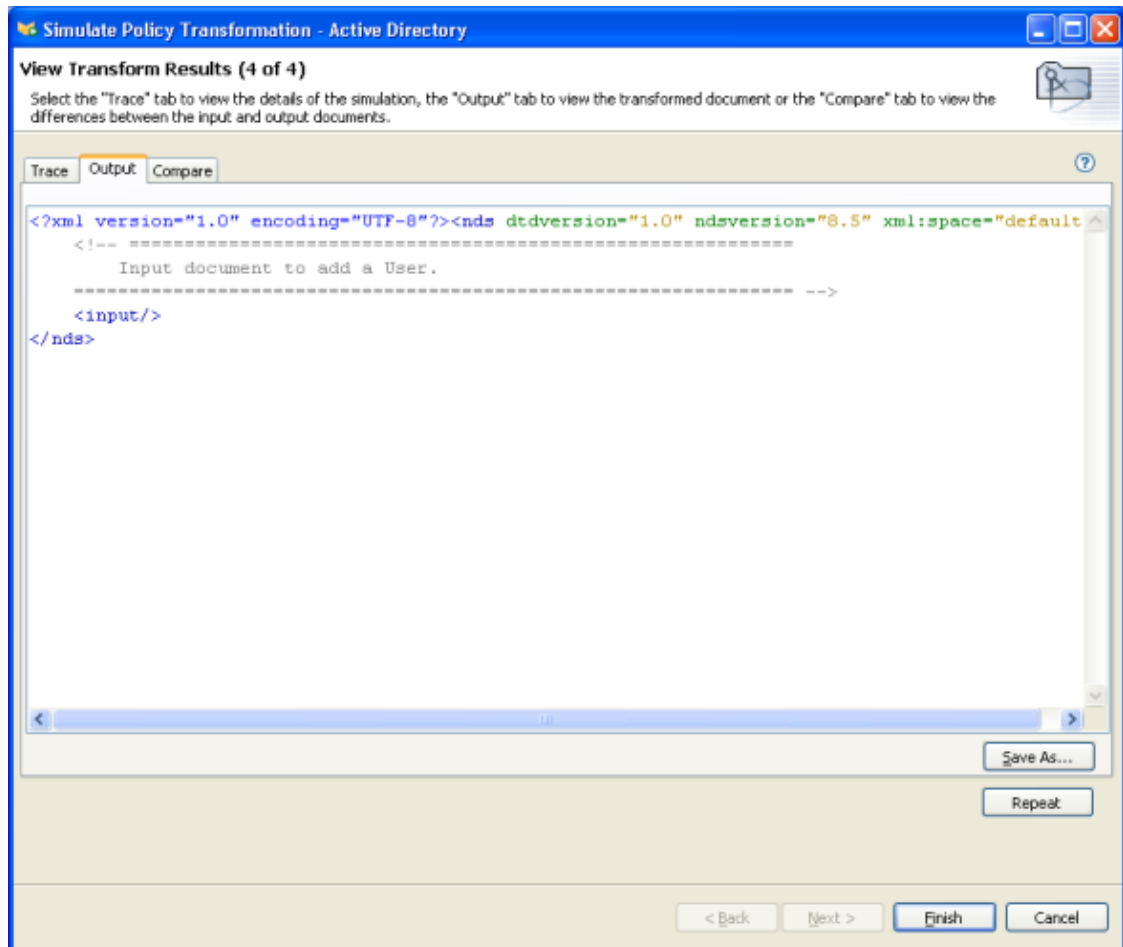
5 Click *Next*.



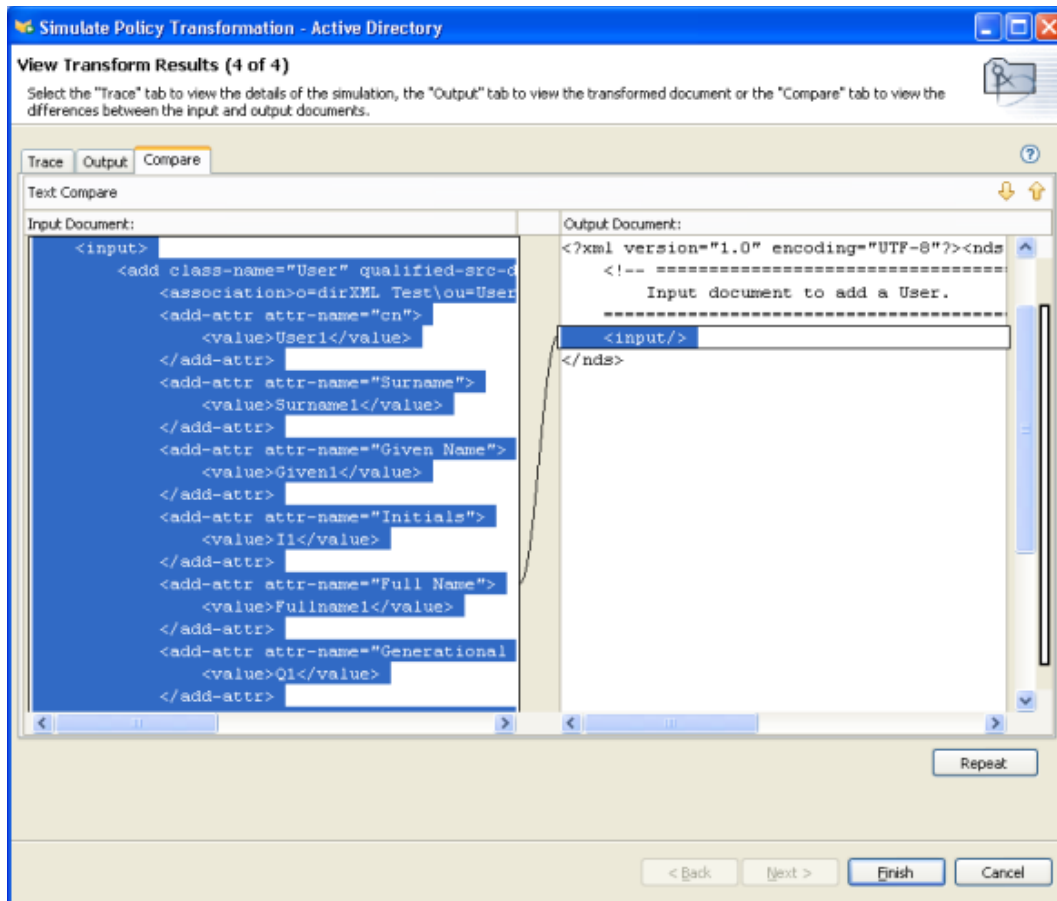
- 6 Select the *Trace* tab to see the results of the event as the policy was processed. The information in this window is the same information that you see in DSTRACE.



7 Select the *Output* tab to see the output document that was generated.



- 8 Select the *Compare* tab to compare the output document to the input document.



- 9 When you are finished looking at the results, click *Repeat* to test another event against the policy.
- 10 When you are finished testing, click *Finish* to close the Policy Simulator.

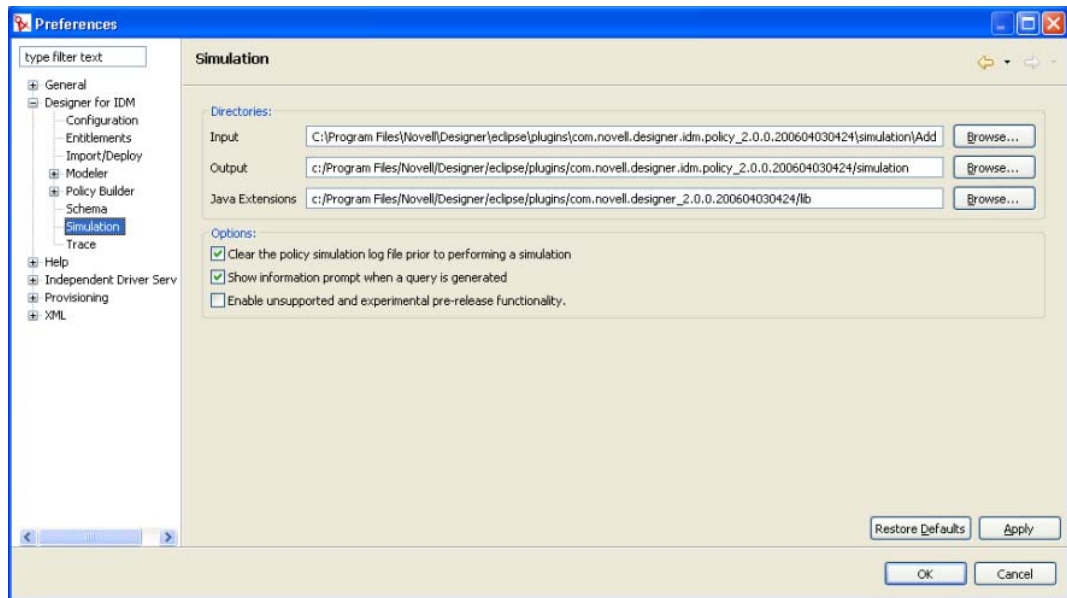
Simulating Policies with Java Extensions

Policies that contain references to external Java extensions can now be simulated by specifying the directory where the jar file is located.

To determine or change the extension directory:

- 1 Select *Windows > Preferences* from the tool bar.
- 2 Navigate to the *Designer for IDM > Simulation* page.

- 3 Copy the jar file containing the Java class to the specified directory and simulate the policy.



NOTE: The *Enable unsupported and experimental pre-release functionality* option enables the Policy Simulator to test the policies against a live Identity Vault or the connected systems. This option is not supported in Designer 1.2 and is not documented.

2.2.8 Editing the DirXML Script

Designer enables you to view, edit, and validate the XML by using an XML editor or text editor.

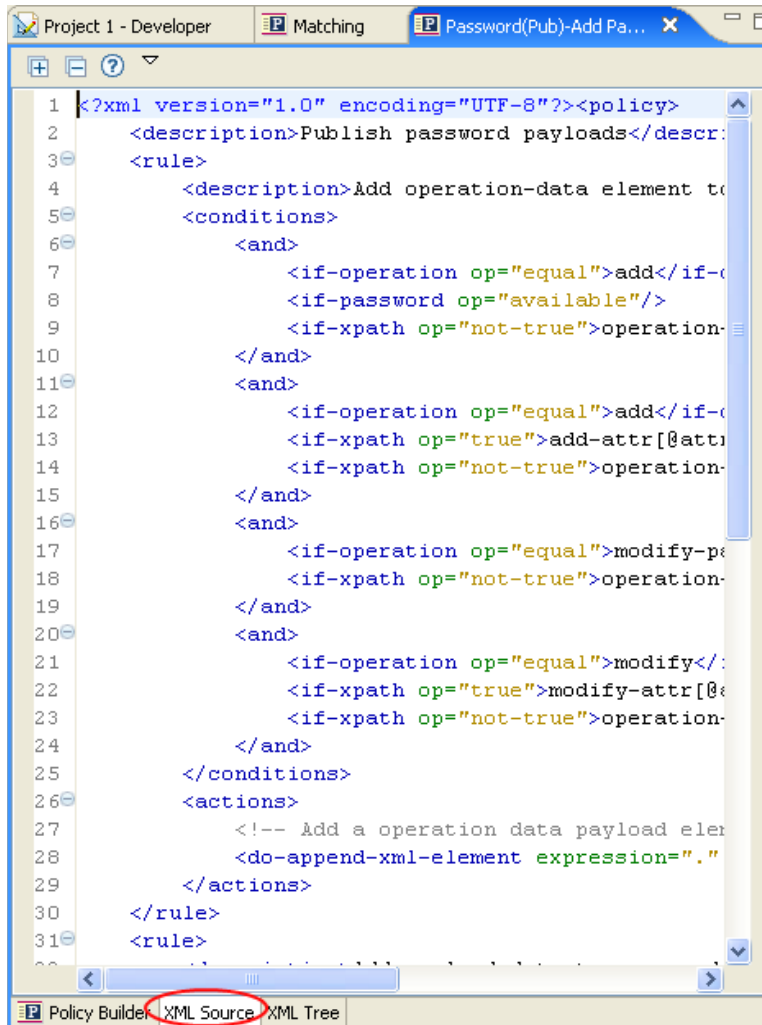
- ♦ [“Viewing the XML Source” on page 114](#)
- ♦ [“Editing the XML Source” on page 118](#)
- ♦ [“Validating the XML Source” on page 120](#)

Viewing the XML Source

You can view the XML Source in XML or in the XML tree format.

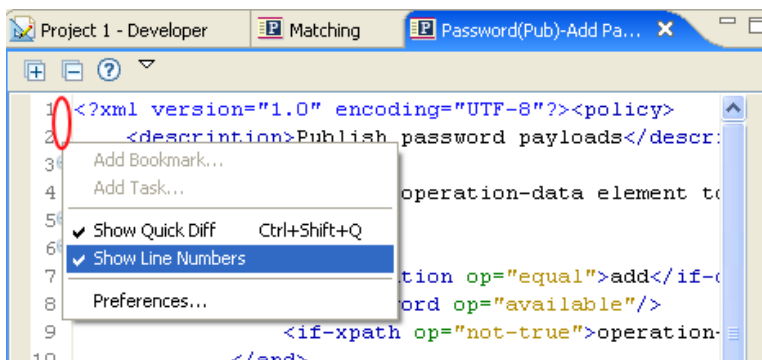
To open the XML Source view:

- 1 Click *XML Source* at the bottom of the Policy Builder's workspace.



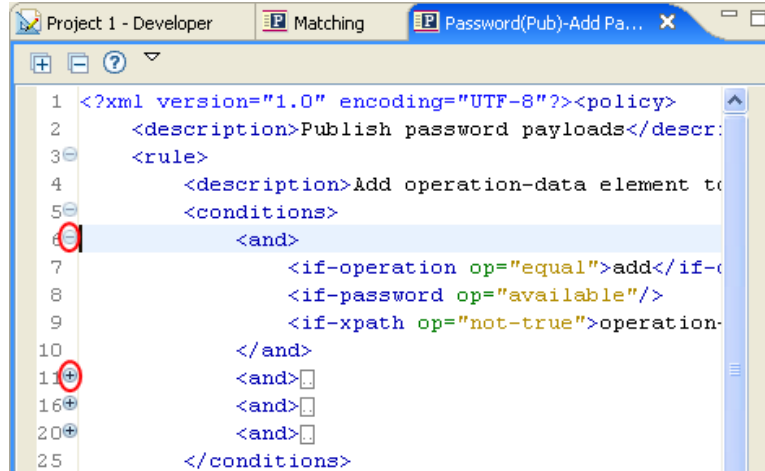
The XML editor displays line numbers.

- 2 To see the line number, right-click in the left margin, then select *Show Line Numbers*.



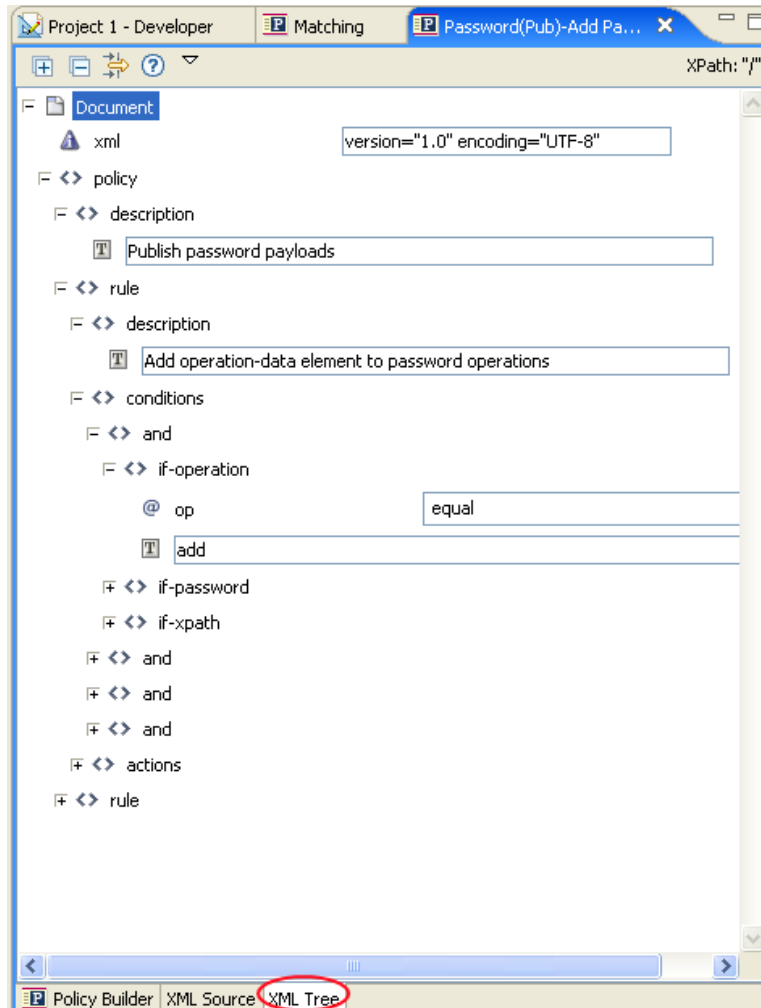
The XML editor expands or collapses the XML by function. If there are functions that contain a large amount of XML, you can collapse the XML by clicking the minus icon in the top left corner.

- 3 To expand all of the XML functions, click the plus icon in the left corner.
Each element has its own plus or minus icon in the left margin.



To view the XML in the tree format:

- 1 Click *XML Tree* at the bottom of the Policy Builder's workspace.

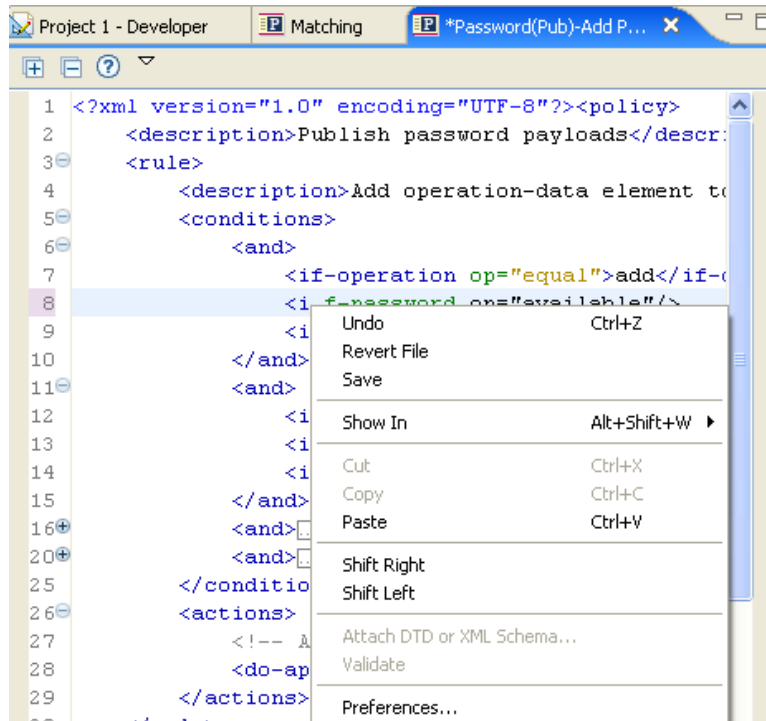


To see the entire tree view, expand each item listed.

Editing the XML Source

You can edit the XML through the XML editor. You can make changes here as well as through the GUI interface.

Figure 2-12 *Editing the XML Source*



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 of the XML Source view is based on the editor that loads.

Right-click to display the list of the functions the XML editor contains.

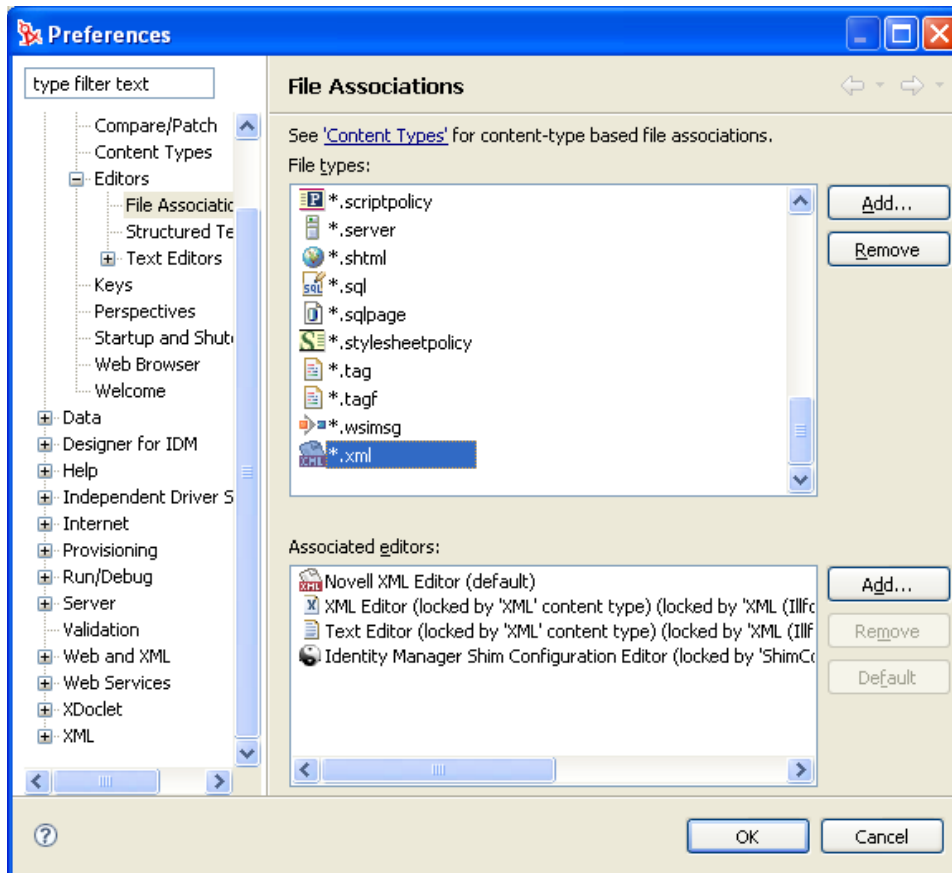
Table 2-5 *XML Editor Options*

Function	Description
<i>Undo</i>	Undoes the last action.
<i>Revert File</i>	Reverts the file to the last version that was saved.
<i>Saves</i>	Saves the file.
<i>Cut</i>	Cuts the selected information.
<i>Paste</i>	Pastes the information into the document.
<i>Shift Right</i>	Indents the line to the right.
<i>Shift Left</i>	Indents the line to the left.
<i>Attach DTD or XML Schema</i>	Attaches a DTD or XML schema file for validation of the policy.

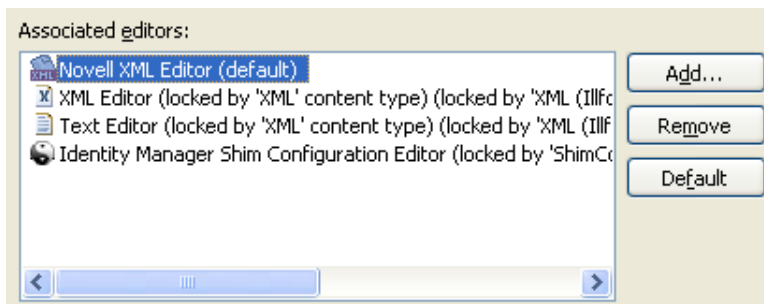
Function	Description
<i>Validate</i>	Validates the XML code.
<i>Preferences</i>	Sets the preferences for the XML editor.

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 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.)

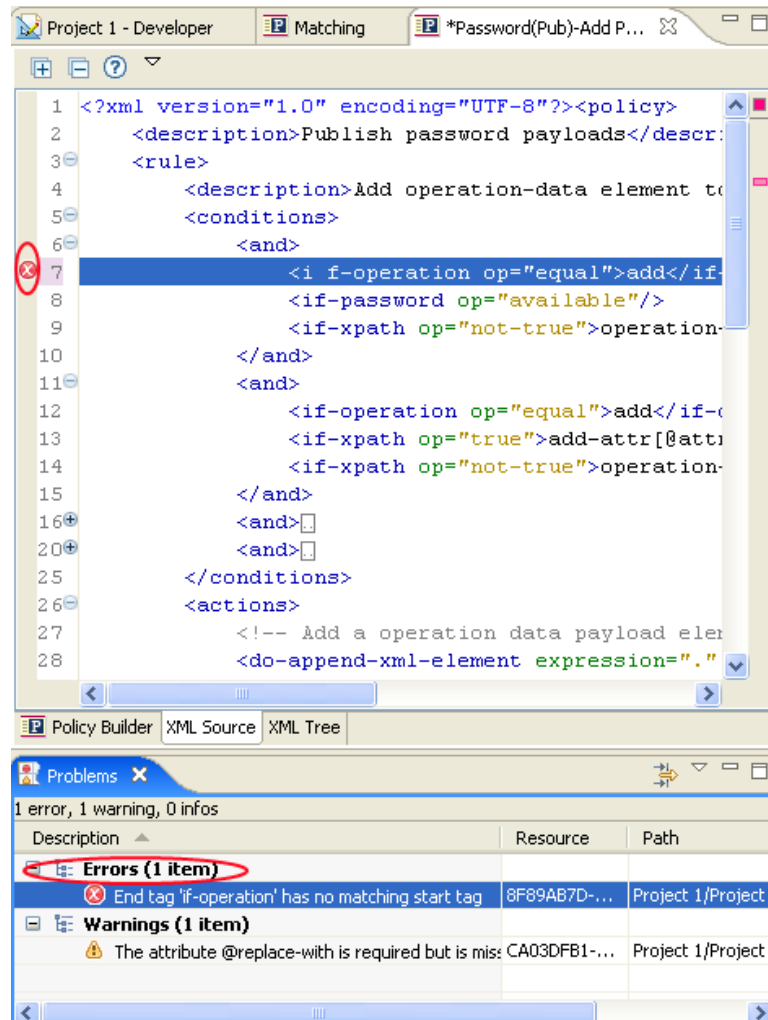


- 5 Click *OK*.
- 6 Close and reopen the Policy Builder.

Validating the XML Source

The XML editor validates the XML code. Right-click, then select *Validate*. If there are errors, a red x is displayed on the line where the error occurs. An explanation at the bottom of the window gives more information about the problem.

Figure 2-13 Validating the XML Source



In this example, the end tag for if-operation has no matching start tag.

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.

Table 2-6 *Common Regular Expressions*

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.
(?u)	Enables Unicode-aware case folding. This flag can impact performance.
(?i)	Enables case-insensitive matching.

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 122](#)
- ◆ [Section 2.5.2, “If Attribute,” on page 123](#)
- ◆ [Section 2.5.3, “If Class Name,” on page 124](#)
- ◆ [Section 2.5.4, “If Destination Attribute,” on page 125](#)
- ◆ [Section 2.5.5, “If Destination DN,” on page 127](#)
- ◆ [Section 2.5.6, “If Entitlement,” on page 127](#)
- ◆ [Section 2.5.7, “If Global Configuration Value,” on page 128](#)
- ◆ [Section 2.5.8, “If Local Variable,” on page 129](#)
- ◆ [Section 2.5.9, “If Named Password,” on page 131](#)
- ◆ [Section 2.5.10, “If Operation,” on page 132](#)
- ◆ [Section 2.5.11, “If Operation Attribute,” on page 133](#)
- ◆ [Section 2.5.12, “If Operation Property,” on page 135](#)
- ◆ [Section 2.5.13, “If Password,” on page 136](#)
- ◆ [Section 2.5.14, “If Source Attribute,” on page 136](#)
- ◆ [Section 2.5.15, “If Source DN,” on page 137](#)
- ◆ [Section 2.5.16, “If XPath Expression,” on page 138](#)

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.
available	There is a non-empty association value specified by the current operation.

Operator	Condition is met when...
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.

Condition: association [?] Operator: * available

OK Cancel * Required

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 208](#).

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.

Operator	Condition is met when...
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 screenshot shows a policy configuration window titled "Filter events: From Users sub-tree, Users not disabled, no consultants or sales people". Under the "Conditions" tab, there is a "Condition Group 1" containing three conditions:

- if source DN not in subtree "Users"
- Or if attribute 'Login Disabled' equal "True"
- Or if attribute 'Title' match ".*consultant|sales.*"

Under the "Actions" tab, the action "veto()" is listed.

The dialog box for configuring the 'If Attribute' condition shows the following fields:

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

Buttons for "OK" and "Cancel" are visible at the bottom left, and a red asterisk with the text "* Required" is at the bottom right.

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

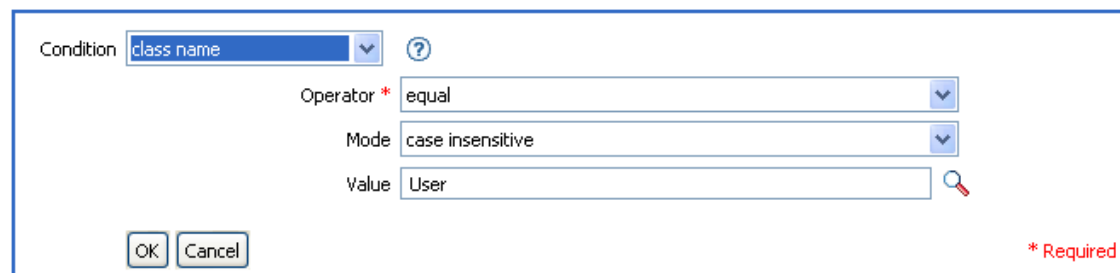
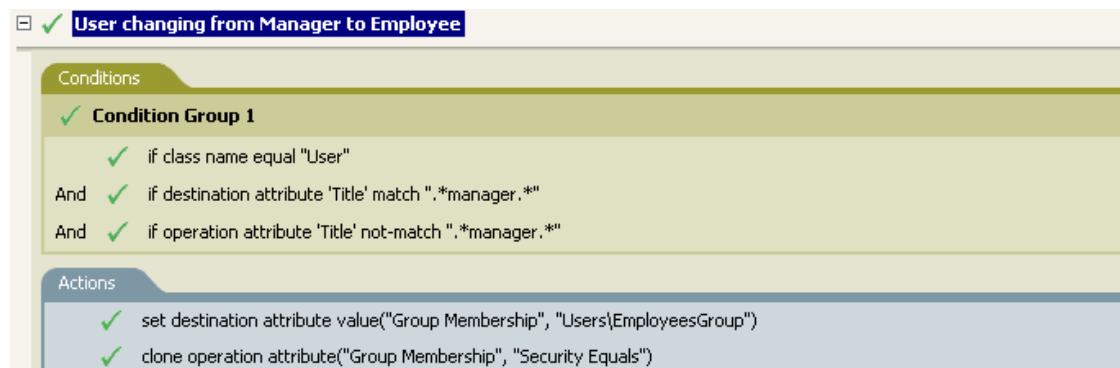
Select the comparison mode. See [“Comparison Modes” on page 208](#).

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.

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 208](#).

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

- Conditions:** A "Condition Group 1" is defined with three conditions:
 - if class name equal "User"
 - And if destination attribute 'Title' match ".*manager.*"
 - And if operation attribute 'Title' not-match ".*manager.*"
- Actions:** Two actions are listed:
 - set destination attribute value("Group Membership", "Users\EmployeesGroup")
 - clone operation attribute("Group Membership", "Security Equals")

The screenshot shows a dialog box for configuring a condition. The "Condition" dropdown is set to "destination attribute". The fields are as follows:

- Name ***: Title
- Operator ***: equal
- Mode**: regular expression
- Value**: .*manager.*

Buttons for "OK" and "Cancel" are at the bottom left. A red asterisk and the text "* Required" are at the bottom right.

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

* Required

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 208](#).

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.
not-equal	Equal would return False.

Example

The screenshot shows a configuration dialog box for an entitlement condition. The 'Condition' dropdown is set to 'entitlement'. The 'Name' field is 'notes-group'. The 'Operator' dropdown is set to 'changing from'. The 'Mode' dropdown is set to 'case insensitive'. The 'Value' field is 'Users'. There are 'OK' and 'Cancel' buttons at the bottom left. A red asterisk and the text '* Required' are at the bottom right.

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 208](#).

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 a dialog box with the following fields and controls:

- Condition:** A dropdown menu set to "global configuration value" with a help icon (?) to its right.
- Name *:** A text input field containing "myGlobalVariable" with a search icon to its right.
- Operator *:** A dropdown menu set to "available".
- Buttons:** "OK" and "Cancel" buttons at the bottom left.
- Legend:** A red asterisk followed by the text "* Required" at the bottom right.

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 208](#).

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.
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](#).

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

- Set local variables to test existence of groups and for placement** (expanded)
- Create ManagersGroup, if needed** (expanded)
- Conditions** section:
 - Condition Group 1** (expanded)
 - if local variable 'manager-group-info' available
 - And if local variable 'manager-group-info' not equal "group"
- Actions** section:
 - add destination object(class name="Group", when="before", dn(Local Variable("manager-group-dn")))
- Create EmployeesGroup, if needed** (collapsed)
- If Title indicates Manager, add to ManagerGroup and set rights** (collapsed)
- If Title does not indicate Manager, add to EmployeeGroup and set rights** (collapsed)

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 * manager-group-info

Operator * not equal

Mode case insensitive

Value group

OK Cancel

* Required

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: ?

Name * 🔍

Operator *

OK Cancel * Required

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.

Value

The values are the operations that the Metadirectory engine looks for in this condition:

- ◆ add
- ◆ add-association
- ◆ check-object-password
- ◆ delete
- ◆ get-named-password
- ◆ modify
- ◆ modify-association
- ◆ modify-password
- ◆ move
- ◆ init-params

- ♦ instance

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

- ✓ **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"))))

Condition

Operator *

Value

* Required

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 208](#).

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

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")

If Title does not indicate Manager, add to EmployeeGroup and set rights

Condition: destination attribute

Name *: Title

Operator *: equal

Mode: regular expression

Value: *.manager.*

OK Cancel * Required

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 208](#).

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 [?] [v]
Name *: myStoredVariable
Operator *: available [v]
[OK] [Cancel] * Required

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

Condition: password [?] [v]
Operator *: available [v]
[OK] [Cancel] * Required

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 208](#).

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

Condition: source attribute

Name *: OU

Operator *: equal

Mode: case insensitive

Value: Users

OK Cancel

* Required

2.5.15 If Source DN

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

Fields

Operator

Select the condition test type.

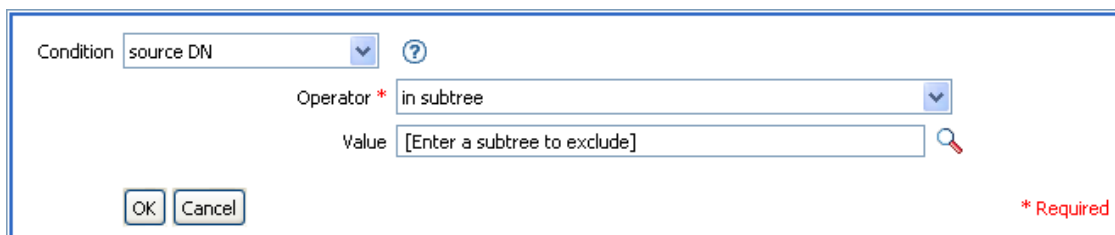
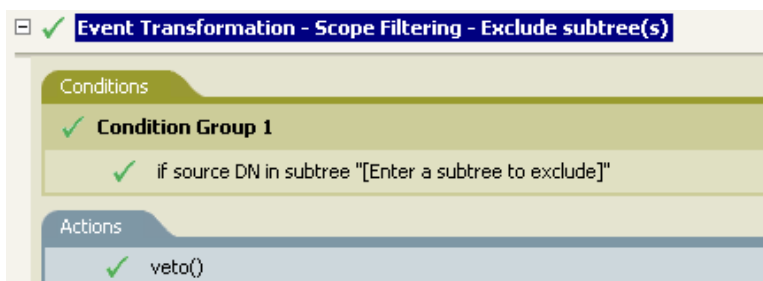
Operator Condition is Met When...

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.

Operator	Condition is met when...
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. For more information, see [“Event Transformation - Scope Filtering - Exclude Subtrees” on page 86](#).



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

Condition ?

Operator *

Value  

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 140](#)
- ◆ [Section 2.6.2, “Add Destination Attribute Value,” on page 141](#)
- ◆ [Section 2.6.3, “Add Destination Object,” on page 142](#)
- ◆ [Section 2.6.4, “Add Source Attribute Value,” on page 144](#)
- ◆ [Section 2.6.5, “Add Source Object,” on page 145](#)
- ◆ [Section 2.6.6, “Append XML Element,” on page 146](#)
- ◆ [Section 2.6.7, “Append XML Text,” on page 147](#)
- ◆ [Section 2.6.8, “Break,” on page 148](#)
- ◆ [Section 2.6.9, “Clear Destination Attribute Value,” on page 148](#)
- ◆ [Section 2.6.10, “Clear Operation Property,” on page 149](#)
- ◆ [Section 2.6.11, “Clear Source Attribute Value,” on page 149](#)
- ◆ [Section 2.6.12, “Clear SSO Credential,” on page 150](#)
- ◆ [Section 2.6.13, “Clone By XPath Expressions,” on page 151](#)
- ◆ [Section 2.6.14, “Clone Operation Attribute,” on page 151](#)
- ◆ [Section 2.6.15, “Delete Destination Object,” on page 152](#)
- ◆ [Section 2.6.16, “Delete Source Object,” on page 153](#)
- ◆ [Section 2.6.17, “Find Matching Object,” on page 153](#)
- ◆ [Section 2.6.18, “For Each,” on page 155](#)
- ◆ [Section 2.6.19, “Generate Event,” on page 156](#)
- ◆ [Section 2.6.20, “Implement Entitlement,” on page 158](#)
- ◆ [Section 2.6.21, “Move Destination Object,” on page 159](#)
- ◆ [Section 2.6.22, “Move Source Object,” on page 160](#)
- ◆ [Section 2.6.23, “Reformat Operation Attribute,” on page 161](#)
- ◆ [Section 2.6.24, “Remove Association,” on page 162](#)
- ◆ [Section 2.6.25, “Remove Destination Attribute Value,” on page 163](#)
- ◆ [Section 2.6.26, “Remove Source Attribute Value,” on page 164](#)
- ◆ [Section 2.6.27, “Rename Destination Object,” on page 165](#)

- ◆ Section 2.6.28, “Rename Operation Attribute,” on page 165
- ◆ Section 2.6.29, “Rename Source Object,” on page 166
- ◆ Section 2.6.30, “Send Email,” on page 166
- ◆ Section 2.6.31, “Send Email From Template,” on page 168
- ◆ Section 2.6.32, “Set Default Attribute Value,” on page 169
- ◆ Section 2.6.33, “Set Destination Attribute Value,” on page 170
- ◆ Section 2.6.34, “Set Destination Password,” on page 171
- ◆ Section 2.6.35, “Set Local Variable,” on page 172
- ◆ Section 2.6.36, “Set Operation Association,” on page 173
- ◆ Section 2.6.37, “Set Operation Class Name,” on page 174
- ◆ Section 2.6.38, “Set Operation Destination DN,” on page 174
- ◆ Section 2.6.39, “Set Operation Property,” on page 175
- ◆ Section 2.6.40, “Set Operation Source DN,” on page 176
- ◆ Section 2.6.41, “Set Operation Template DN,” on page 176
- ◆ Section 2.6.42, “Set Source Attribute Value,” on page 177
- ◆ Section 2.6.43, “Set Source Password,” on page 178
- ◆ Section 2.6.44, “Set SSO Credential,” on page 179
- ◆ Section 2.6.45, “Set SSO Passphrase,” on page 179
- ◆ Section 2.6.46, “Set XML Attribute,” on page 180
- ◆ Section 2.6.47, “Status,” on page 181
- ◆ Section 2.6.48, “Strip Operation Attribute,” on page 182
- ◆ Section 2.6.49, “Strip XPath,” on page 182
- ◆ Section 2.6.50, “Trace Message,” on page 183
- ◆ Section 2.6.51, “Veto,” on page 184
- ◆ Section 2.6.52, “Veto If Operational Attribute Not Available,” on page 185

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

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

- A dropdown menu labeled "Do" with the value "add association" and a help icon (?).
- A dropdown menu labeled "Select mode:" with the value "add to current operation" and a help icon (?).
- A dashed box containing an information icon (i) and the text "Leave the DN field below blank to use the current object".
- A text input field labeled "Enter DN:" containing the value "Source DN()".
- A text input field labeled "Enter association: *" containing the value "Source Name()".
- Two buttons at the bottom: "OK" and "Cancel".

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. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2”](#) on page 76.

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="Organizational Unit", 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="organizational Unit", 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: * 📄

* Required

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 141](#) 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. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2” on page 76](#).

The screenshot shows a policy rule configuration window titled "Command Transformation - Create Departmental Container - Part 1". It is divided into two main sections: "Conditions" and "Actions".

- Conditions:** A "Condition Group 1" is defined with a single condition: "if operation equal 'add'".
- Actions:** Three actions are listed:
 - set local variable("target-container", Destination DN(length="-2"))
 - set local variable("does-target-exist", Destination
 - Attribute("objectclass", class name="Organizational Unit", dn(Local Variable("target-container"))))

The screenshot shows a policy rule configuration window titled "Command Transformation - Create Departmental Container - Part 2". It is divided into two main sections: "Conditions" and "Actions".

- Conditions:** A "Condition Group 1" is defined with two conditions:
 - if local variable 'does-target-exist' available
 - And if local variable 'does-target-exist' equal ""
- Actions:** Two actions are listed:
 - add destination object(class name="organizational Unit", 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") 📅

OK Cancel * Required

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


Do **add source attribute value** 


Enter attribute name: * 

Enter class name: 

Select object: 

Enter association: * 

Enter value type: 

Enter string: * 

* Required

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 \(page 144\)](#) 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

✓ add source object(class name="User", dn("Users\John Smith"))

Do ?

Enter class name: * 🔍

Enter DN: * 📄

* Required

✓ add source attribute value("Title", class name="User", "Manager")

Do ?

Enter attribute name: * 🔍

Enter class name: 🔍

Select object: ▾

Enter value type: ▾

Enter string: * 📄

* Required

2.6.6 Append XML Element

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

Fields

Variable 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

✓ append XML element("jdbc:sql", "../jdbc:statement[last()])

Do ?

Enter variable name: * 🔍

Enter XPath expression: * 🏠 ➡

* Required

✓ append XML text("../jdbc:statement[last()]/jdbc:sql", "UPDATE dixml.emp SET fname"+Operation Attribute("Member"))

Do ?

Enter XPath expression: * 🏠 ➡

Enter string: * 📄

* Required

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

✓ append XML element("jdbc:sql", "../jdbc:statement[last()])

Do ?

Enter variable name: * 🔍

Enter XPATH expression: * 🗑️ ↗️

* Required

✓ append XML text("../jdbc:statement[last()]/jdbc:sql", "UPDATE dixml.emp SET frame"+Operation Attribute("Member"))

Do ?

Enter XPATH expression: * 🗑️ ↗️

Enter string: * 🗑️

* Required

2.6.8 Break

Ends processing of the current operation by the current policy.

Example

Do ?

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

Do 

Enter attribute name: * 

Enter class name: 

Select mode:

Select object:

* Required

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

Do 

Enter property name: *

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

Do 

Enter attribute name: * 

Enter class name: 

Select object: 

* Required

2.6.12 Clear SSO Credential

Clears the Single Sign On credential, so objects can be deprovisioned. This action is part of the Credential Provisioning policies. For more information, see [Chapter 4, “Novell Credential Provisioning Policies,” on page 327](#).

Fields

Credential Store Object DN

Specify the DN of the repository object.

Target User DN

Specify the DN of the target users.

Application Credential ID

Specify the application credential that is stored in the application object.

Login Parameter Strings

Specify each login parameter for the application. The login parameters are the authentication keys stored in the application object.

Example

Do ?

Enter credential store object DN: * 🔍

Render browsed DN relative to policy

Enter target user DN: * 📄

[Populate the following from an application object](#)

Enter application credential ID: *

Enter login parameter strings: 📄

* Required

2.6.13 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

Do ?

Enter source XPATH expression: * 📄 🖱️

Enter destination XPATH expression: * 📄 🖱️

* Required

2.6.14 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:

- Checklist:**
 - 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:**
 - 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")
- If Title does not indicate Manager, add to EmployeeGroup and set rights**

A detailed view of the 'clone operation attribute' action is shown in a separate window:

- Do:** clone operation attribute
- Enter source name:** * Group Membership
- Enter destination name:** Security Equals
- Buttons:** OK, Cancel
- Footer:** * Required

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.

2.6.15 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 with a title bar. Inside, there is a dropdown menu labeled 'Do' with the text 'delete destination object' and a question mark icon. Below it are two more dropdown menus: 'Select mode:' with 'add to current operation' and 'Select object:' with 'Current object'. At the bottom left are 'OK' and 'Cancel' buttons. At the bottom right is a red asterisk followed by the text '* Required'.

2.6.16 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 with a title bar. Inside, there is a dropdown menu labeled 'Do' with the text 'delete source object' and a question mark icon. Below it are two input fields: 'Select object:' with a dropdown menu showing 'DN' and 'Enter DN: *' with a text input field containing '"Uses\John Smith"' and a list icon. At the bottom left are 'OK' and 'Cancel' buttons. At the bottom right is a red asterisk followed by the text '* Required'.

2.6.17 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.

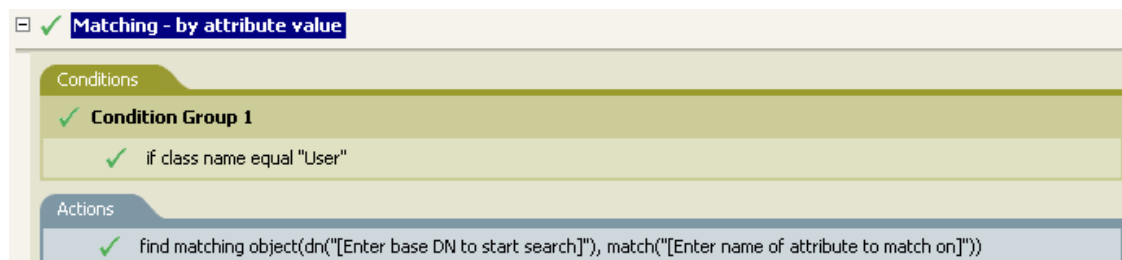
The DN argument is required when scope is “entry”, and is optional otherwise. At least one match attribute is required when scope is “subtree” or “subordinates”.

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 . If multiple results are returned, then the destination DN of the current operation is set to the single character .

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. For more information, see [“Matching - By Attribute Value” on page 94](#).



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.

2.6.18 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 158 action.

Example

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

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()

OK Cancel

* Required

2.6.19 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.
target-type	Integer specifying a predefined format for the target. Predefined values for target-type are currently: <ul style="list-style-type: none"> ◆ 0 = None ◆ 1 = Slash Notation ◆ 2 = Dot Notation ◆ 3 = LDAP Notation
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-].*"

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 generate event

Enter ID: * 1000

Select level: informational

Enter strings: text1

OK Cancel * Required

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

Name String Value

text1 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 reads User:jsmith added to the Training\Users\Active\Users1 container.

2.6.20 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

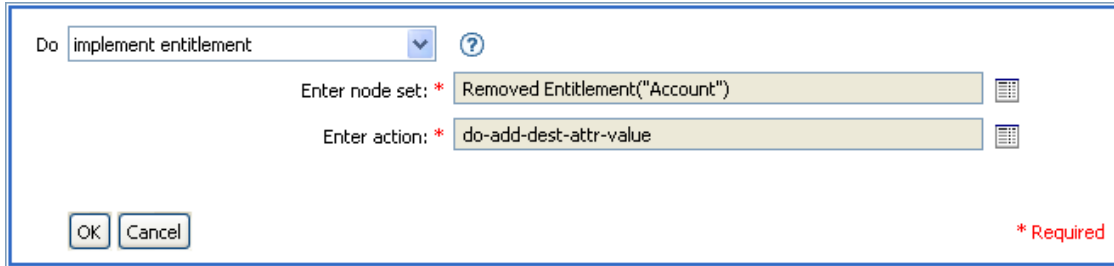
Node Set

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

Action

Actions that implement the specified entitlements.

Example



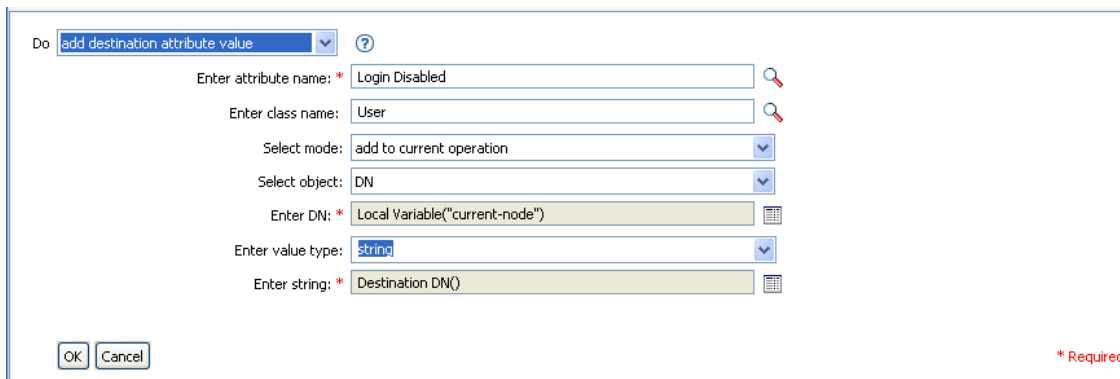
Do: implement entitlement

Enter node set: * Removed Entitlement("Account")

Enter action: * do-add-dest-attr-value

OK Cancel * Required

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



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()

OK Cancel * Required

2.6.21 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 displays the configuration for a policy titled "On Termination, disable user and move to Disabled container". It 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"))

This dialog box is used to configure the "move destination object" action. It includes the following fields:

- 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"

Buttons for "OK" and "Cancel" are located at the bottom left. A red asterisk and the text "* Required" are at the bottom right.

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.

2.6.22 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

Do: move source object

Select object to move: Current object

Select container to move to: DN

Enter DN: * "Users\Inactive"

OK Cancel

* Required

2.6.23 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 construct 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. For more information, see [“Input or Output Transformation - Reformat Telephone Number from \(nnn\) nnn-nnnn to nnn-nnn-nnnn”](#) on page 88.

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

Conditions

Condition Group 1

Define new condition here

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\\d))s*(\\d\\d\\d\\d)-(\\d\\d\\d\\d)\$", "\$1-\$2-\$3")

OK Cancel

* Required

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.

2.6.24 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. For more information, see [“Command Transformation - Publisher Delete to Disable” on page 78](#).

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()))

Do: remove association

Select mode: add to current operation

Enter association: * Association()

OK Cancel

* Required

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.

2.6.25 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 (dropdown menu)
- Enter attribute name: *** Title (text input)
- Enter class name:** User (text input)
- Select mode:** add to current operation (dropdown menu)
- Select object:** Current object (dropdown menu)
- Enter value type:** string (dropdown menu)
- Enter string: *** Destination DN() (text input)

Buttons: OK, Cancel. A red asterisk indicates required fields. A red asterisk and the text '* Required' are located at the bottom right of the dialog.

2.6.26 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 dialog box is titled 'remove source attribute value'. It contains the following fields:

- Do: remove 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: * Destination DN() (text input)

Buttons: OK, Cancel. A red asterisk indicates required fields. A red text '* Required' is located at the bottom right.

2.6.27 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 dialog box is titled 'rename destination object'. It contains the following fields:

- Do: rename destination object (dropdown menu)
- Select mode: add to current operation (dropdown menu)
- Select object: DN (dropdown menu)
- Enter DN: * Users\John Smith (text input)
- Enter string: * Johnny (text input)

Buttons: OK, Cancel.

2.6.28 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 "Do" with a dropdown menu set to "rename operation attribute" and a help icon. Below the dropdown are two input fields: "Enter source name: * Surname" and "Enter destination name: sn". Both input fields have a magnifying glass icon to their right. At the bottom left are "OK" and "Cancel" buttons. At the bottom right is the text "* Required".

2.6.29 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

The screenshot shows a dialog box titled "Do" with a dropdown menu set to "rename source object" and a help icon. Below the dropdown are three input fields: "Select object: DN" (with a dropdown arrow), "Enter DN: * \"Users\John Smith\"" (with a list icon), and "Enter string: * \"Johnny\"" (with a list icon). At the bottom left are "OK" and "Cancel" buttons. At the bottom right is the text "* Required".

2.6.30 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.
message	Specifies the content of the e-mail message.
encoding	Specifies the character encoding to use for the e-mail message.

Example

Do: send email

Enter ID: user

Enter server: * smtp.company.com

Enter password: ●●●●●●●●

Select message type: text

Enter strings: to, cc, bcc, from, subject, message

OK Cancel

* Required

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"

2.6.31 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 dialog box is titled "Do" and contains the following fields:

- A dropdown menu with the selected value "send email from template".
- A text input field for "Enter notification DN: *" with the value "Security\Default Notification Collection".
- A text input field for "Enter template DN: *" with the value "Security\Default Notification Collection>Password Set Fail".
- A text input field for "Enter password:" which is currently empty.
- A text input field for "Enter strings:" with the value "to, cc".

At the bottom left are "OK" and "Cancel" buttons. At the bottom right is a red asterisk followed by the text "* Required".

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

Name	String Value
to	"to_user1@company.com"
cc	"cc_user@company.com"

2.6.32 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. For more information, see [“Creation - Set Default Attribute Value” on page 82](#).

The window shows a rule configuration with the following structure:

- Conditions**
 - Condition Group 1
 - if class name equal "User"
- Actions**
 - set default attribute value("[Enter attribute name]", write-back="true", "[Enter default attribute value]")

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 67](#) 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.

2.6.33 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. For more information, see [“Command Transformation - Publisher Delete to Disable”](#) on page 78.

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()))

The screenshot shows the configuration dialog for the "set destination attribute value" action. The dialog includes the following fields:

- 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"

Buttons for "OK" and "Cancel" are at the bottom left. A red asterisk and the text "* Required" are at the bottom right.

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 64 for more information about the builder.

2.6.34 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

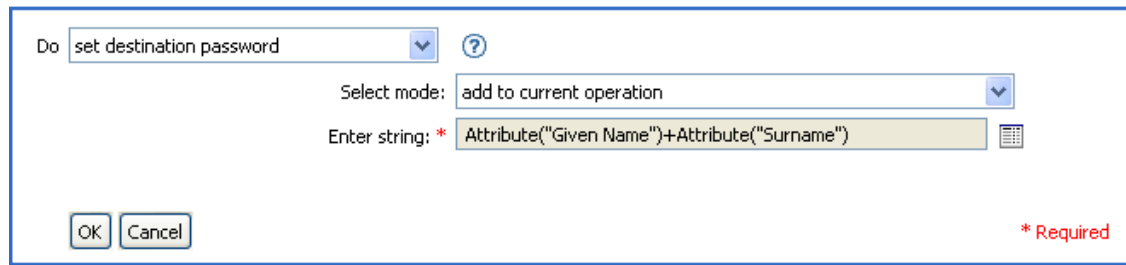
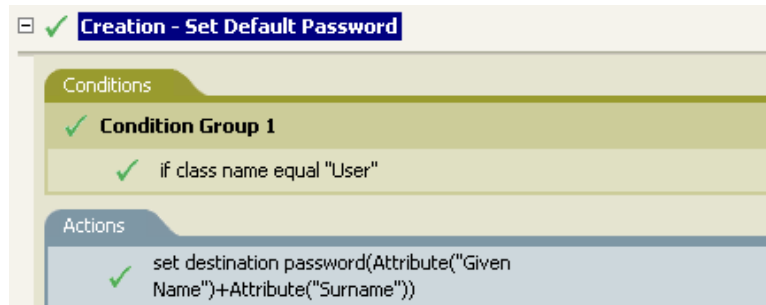
Select the target object. This object can be the current object, or be specified by a DN or an association.

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. For more information, see [“Creation - Set Default Password” on page 83](#).



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

2.6.35 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 a title bar that reads "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"))))

The screenshot shows a dialog box for configuring a "set local variable" action. The "Do" dropdown is set to "set local variable".

Enter variable name: * LVUsers1

Select variable type: String

Enter string: * "User:"+Operation Attribute("cn")+ added to the "+"Training\

Buttons: OK, Cancel

* Required

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 64](#) for more information.

2.6.36 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()

OK Cancel * Required

2.6.37 Set Operation Class Name

Sets the object class name for the current operation.

Fields

String

Provide the new class name.

Example

Do set operation class name ?

Enter string: * "User"

OK Cancel * Required

2.6.38 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. For more information, see [“Creation - Set Default Attribute Value” on page 82.](#)

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

2.6.39 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

2.6.40 Set Operation Source DN

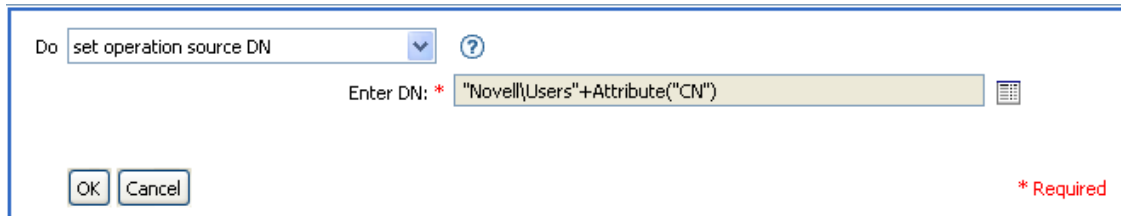
Sets the source DN for the current operation.

Fields

DN

Specify the new source DN.

Example



Do: set operation source DN

Enter DN: * "Novell\Users"+Attribute("CN")

OK Cancel

* Required

2.6.41 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.

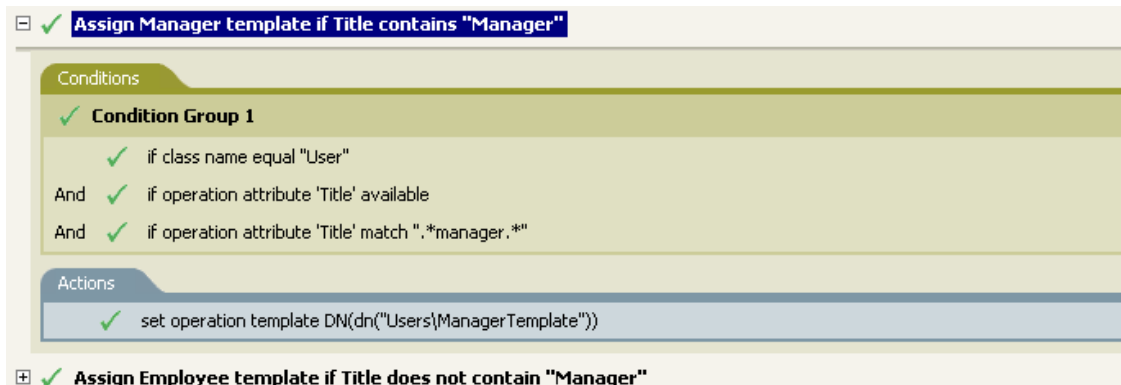
Fields

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 set operation template DN ?

Enter DN: * "Users\ManagerTemplate"

OK Cancel

* Required

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.

2.6.42 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.

The screenshot shows a policy configuration window titled "Push back on email changing". It is divided into two main sections: "Conditions" and "Actions".

- 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")

The screenshot shows the configuration dialog for the "set source attribute value" action. The "Do" dropdown is set to "set source attribute value".

- Enter attribute name: * Email
- Enter class name: (empty)
- Select object: Current object
- Enter value type: string
- Enter string: * Destination Attribute("Internet EMail Address")

Buttons: OK, Cancel. A red asterisk indicates required fields.

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

2.6.43 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 the configuration dialog for the "set source password" action. The "Do" dropdown is set to "set source password".

- Enter string: * Attribute("Given Name")+Attribute("Surname")

Buttons: OK, Cancel. A red asterisk indicates required fields.

2.6.44 Set SSO Credential

Sets the SSO credential when a user object is created or when a password is modified. This action is part of the Credential Provisioning policies. For more information, see [Chapter 4, “Novell Credential Provisioning Policies,” on page 327.](#)

Fields

Credential Store Object DN

Specify the DN of the repository object.

Target User DN

Specify the DN of the target users.

Application Credential ID

Specify the application credential that is stored in the application object.

Login Parameter Strings

Specify each login parameter for the application. The login parameters are the authentication keys stored in the application object.

Example

Do set SSO credential ?

Enter credential store object DN: * Novell\Driver Set\GroupWise\GroupWise_Repository 🔍

Render browsed DN relative to policy

Enter target user DN: * Destination Attribute("DirXML-ADContext", class name="User"); 📄

[Populate the following from an application object](#)

Enter application credential ID: * GroupWise_Credential

Enter login parameter strings: Username, Password 📄

OK Cancel * Required

2.6.45 Set SSO Passphrase

Sets the Novell SecureLogin[®] passphrase and answer when a User object is provisioned. This action is part of the Credential Provisioning policies. For more information, see [Chapter 4, “Novell Credential Provisioning Policies,” on page 327.](#)

Fields

Credential Store Object DN

Specify the DN of the repository object.


Target User DN


Specify the DN of the target users.

Question and Answer Strings


Specify the SecureLogin passphrase question and answer.


Example

Do 


Enter credential store object DN: * 

Render browsed DN relative to policy

Enter target user DN: * 

Enter question and answer strings: 

* Required

The SecureLogin passphrase question and answer are stored as strings in the policy. Click the *Edit the strings* icon  to launch the string builder. Specify the passphrase question and answer.

2.6.46 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 the Policy Builder interface, showing the configuration of 'set XML attribute' actions. Each screenshot is a dialog box with a title bar containing a green checkmark and the XML expression: `set XML attribute("cert-id", ".", "c:\lotus\domino\data\eng.id")`.

The first screenshot shows the configuration for the variable `cert-id`. The 'Do' dropdown is set to 'set XML attribute'. The 'Enter variable name' field contains `cert-id`. The 'Enter XPATH expression' field contains `.`. The 'Enter string' field contains `"c:\lotus\domino\data\eng.id"`. There are 'OK' and 'Cancel' buttons at the bottom left, and a red asterisk with the text '* Required' at the bottom right.

The second screenshot shows the configuration for the variable `cert-pwd`. The 'Do' dropdown is set to 'set XML attribute'. The 'Enter variable name' field contains `cert-pwd`. The 'Enter XPATH expression' field contains `.`. The 'Enter string' field contains `"certify2eng"`. There are 'OK' and 'Cancel' buttons at the bottom left, and a red asterisk with the text '* Required' at the bottom right.

2.6.47 Status

Generates a status notification.

Fields

Level

Specify the status level of the notification.

Message

Provide the status message by using the Argument Builder.

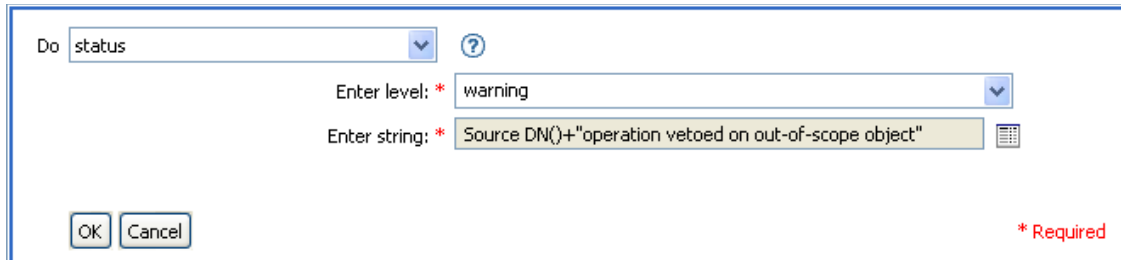
Remarks

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



Do status

Enter level: * warning

Enter string: * Source DN()+`"operation vetoed on out-of-scope object"`

* Required

2.6.48 Strip Operation Attribute

Strips all occurrences of an attribute from the current operation.

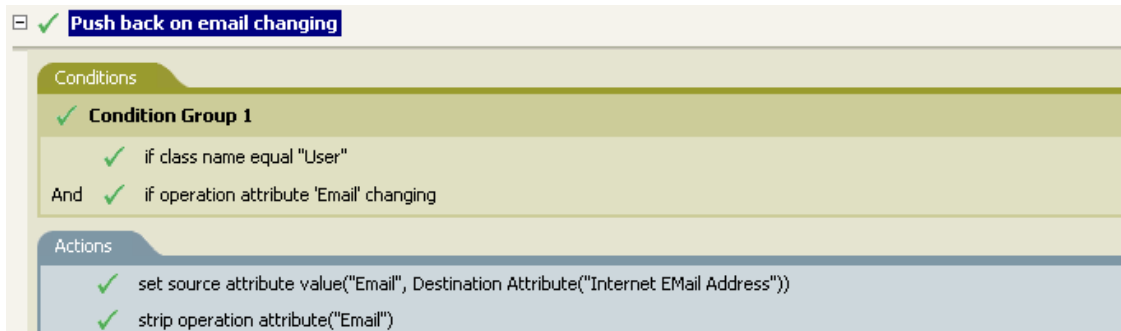
Fields

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.



☐ ✓ **Push back on email changing**

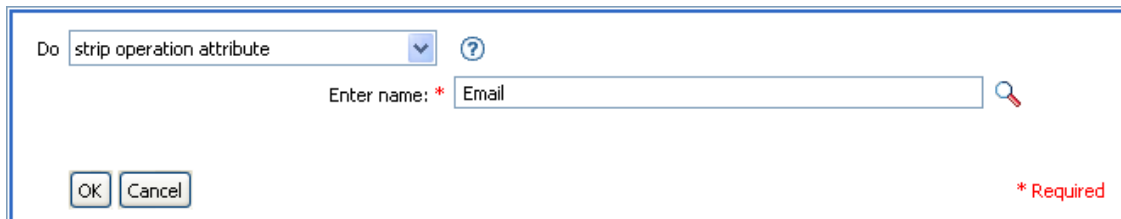
Conditions

✓ **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 strip operation attribute

Enter name: * Email

* Required

The action strips the attribute of Email. The value that is kept is what was in the destination Email attribute.

2.6.49 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

Do ?

Enter XPath expression: *  

* Required

2.6.50 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 3.0.1 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: *

* Required

The action sends a trace message to DSTRACE. The contents of the local variable is LVUsers1 and it shows up in yellow in DSTRACE.

2.6.51 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. For more information, see "[Event Transformation - Scope Filtering - Exclude Subtrees](#)" on page 86.

Event Transformation - Scope Filtering - Exclude subtree(s)

Conditions
 Condition Group 1
 if source DN in subtree "[Enter a subtree to exclude]"

Actions
 veto()

Do veto 

* Required

The action vetoes all events that come from the specified subtree.

2.6.52 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.


User required attributes: First/Last Name, Title, Description, Email


Conditions

- Condition Group 1**
 - if class name equal "User"

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")
- veto if operation attribute not available("Internet EMail Address")

Do veto if operation attribute not available 

Enter name: * 

* Required

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 186](#)
- ◆ [Section 2.7.2, “Association,” on page 187](#)
- ◆ [Section 2.7.3, “Attribute,” on page 187](#)
- ◆ [Section 2.7.4, “Class Name,” on page 188](#)
- ◆ [Section 2.7.5, “Destination Attribute,” on page 188](#)
- ◆ [Section 2.7.6, “Destination DN,” on page 189](#)
- ◆ [Section 2.7.7, “Destination Name,” on page 190](#)
- ◆ [Section 2.7.8, “Entitlement,” on page 190](#)
- ◆ [Section 2.7.9, “Global Configuration Value,” on page 191](#)
- ◆ [Section 2.7.10, “Local Variable,” on page 191](#)
- ◆ [Section 2.7.11, “Named Password,” on page 192](#)
- ◆ [Section 2.7.12, “Operation,” on page 192](#)
- ◆ [Section 2.7.13, “Operation Attribute,” on page 193](#)
- ◆ [Section 2.7.14, “Operation Property,” on page 194](#)
- ◆ [Section 2.7.15, “Password,” on page 194](#)
- ◆ [Section 2.7.16, “Removed Attribute,” on page 194](#)
- ◆ [Section 2.7.17, “Removed Entitlement,” on page 194](#)
- ◆ [Section 2.7.18, “Source Attribute,” on page 195](#)
- ◆ [Section 2.7.19, “Source DN,” on page 195](#)
- ◆ [Section 2.7.20, “Source Name,” on page 196](#)
- ◆ [Section 2.7.21, “Text,” on page 196](#)
- ◆ [Section 2.7.22, “Unique Name,” on page 197](#)
- ◆ [Section 2.7.23, “Unmatched Source DN,” on page 199](#)
- ◆ [Section 2.7.24, “XPath,” on page 200](#)

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")
```

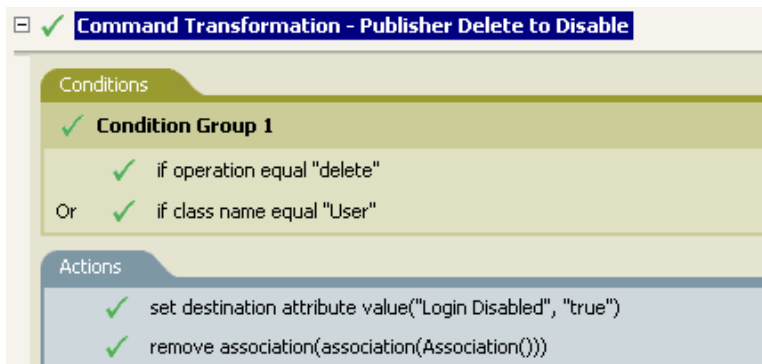
2.7.2 Association

Expands to the association value from the current operation.

Example

The example is from the predefined rules that come with Identity Manager. For more information on the predefined rule, see [“Command Transformation - Publisher Delete to Disable” on page 78](#).

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. For more information, see [“Creation - Set Default Password” on page 83](#).

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

- Condition Group 1
 - if class name equal "User"

Actions

- set destination password(Attribute("Given Name")+Attribute("Surname"))

Attribute("Given Name")

Attribute("Surname")

Editor

Name: *

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
 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


 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 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. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2”](#) on page 76.

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="Organizational Unit", 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("Company Name")

2.7.10 Local Variable

Expands to the value of a local variable.

Fields

Name

Specify the name of the local variable.

Example

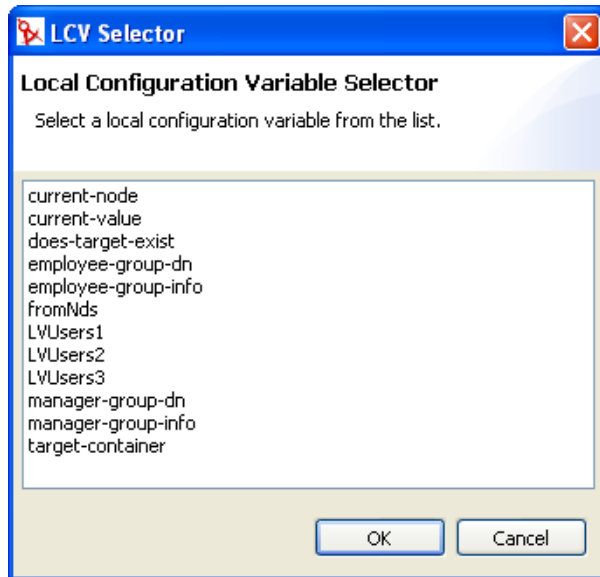
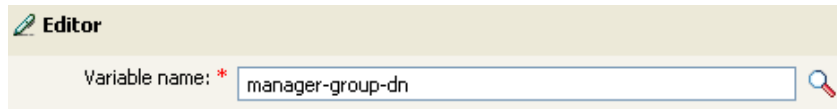
The example is from the Govern Groups for User Based on Title policy which is available for download from Novell's support Web site. For more information, see [“Downloadable Identity Manager Policies” on page 36](#).

The action Add Destination Object uses the Local Variable token.

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

- Set local variables to test existence of groups and for placement** (expanded)
- Create ManagersGroup, if needed** (expanded)
- Conditions** section:
 - Condition Group 1** (expanded)
 - if local variable 'manager-group-info' available
 - And if local variable 'manager-group-info' not equal "group"
- Actions** section:
 - add destination object(class name="Group", when="before", dn(Local Variable("manager-group-dn")))
- Create EmployeesGroup, if needed** (collapsed)
- If Title indicates Manager, add to ManagerGroup and set rights** (collapsed)
- If Title does not indicate Manager, add to EmployeeGroup and set rights** (collapsed)

..... 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 the specified attribute from the current XDS operation. It is different from Source Attribute and Destination Attribute, because it is always accessed directly from what is available in the current XDS operation as opposed to being queried from the source or destination data stores. It does not include the removed values from a modify operation.

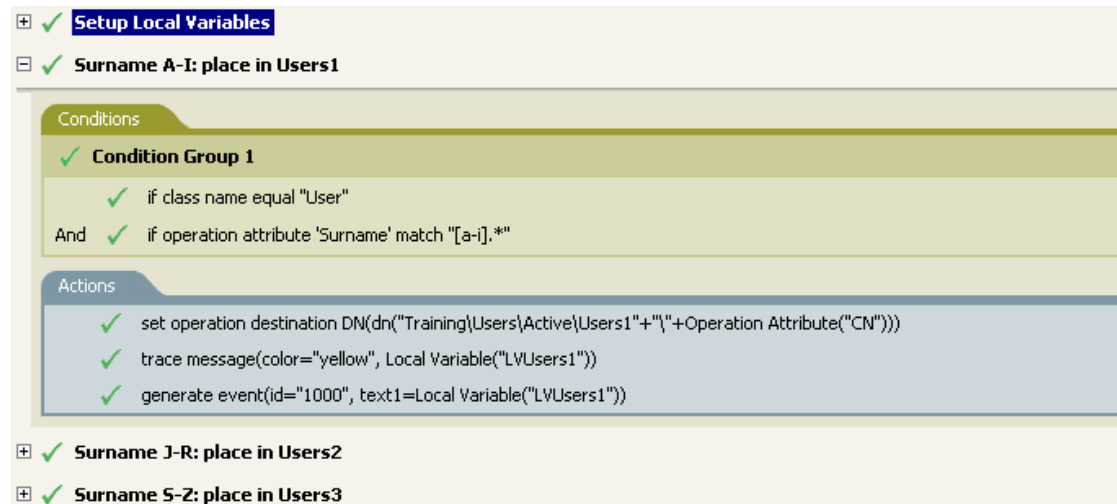
Fields

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 shows a list of rules in the Policy Builder. The first rule is expanded to show its configuration:

- Setup Local Variables** (checked)
- Surname A-I: place in Users1** (checked)
 - Conditions**
 - Condition Group 1** (checked)
 - if class name equal "User" (checked)
 - And if operation attribute 'Surname' match "[a-].*" (checked)
 - Actions**
 - set operation destination DN(dn("Training\Users\Active\Users1"+"")+Operation Attribute("CN")) (checked)
 - trace message(color="yellow", Local Variable("LVUsers1")) (checked)
 - generate event(id="1000", text1=Local Variable("LVUsers1")) (checked)
- Surname J-R: place in Users2** (checked)
- Surname S-Z: place in Users3** (checked)

 "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.

2.7.14 Operation Property

The XML attribute attached to an <operation-data> element by a policy. It is a place for policies to store and forward information for consumption by other policies.

Remarks

An XML attribute is a name value pair associated with an element in the XDS document.

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("CN", class name="User")

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". It is divided into two main sections: "Conditions" and "Actions".

Conditions:

- Condition Group 1:** Contains one condition: "if class name equal 'User'".
- And:** A logical connector between the two condition groups.
- Condition Group 2:** Contains two conditions: "if operation equal 'add'" and "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"))))

..... "Users\ManagersGroup"

The screenshot shows the "Editor" window for a text token. The text "Users\ManagersGroup" is entered into the text field. A magnifying glass icon is visible on the right side of the text field.

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.

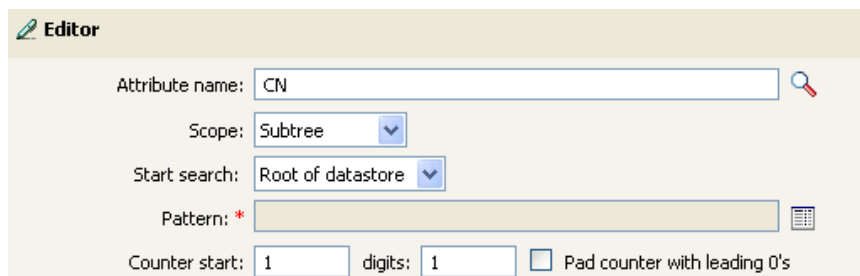
The counter can be set to start at a different number using the counter start field. The counter uses the number of digits specified by the digits field. If the number of digits is less than those specified, then the counter is right padded with zeros. When the number of digits exceeds those specified, 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", Lower Case()+Attribute("Surname"))

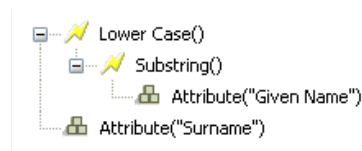
The following is an example of the Editor pane when constructing the unique name argument:



The screenshot shows the 'Editor' pane with the following fields and values:

- 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. For more information, see [“Matching - Subscriber Mirrored - LDAP Format” on page 92](#).

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.

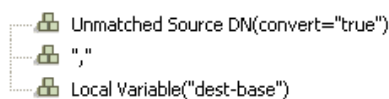
Matching - Subscriber Mirrored - LDAP format

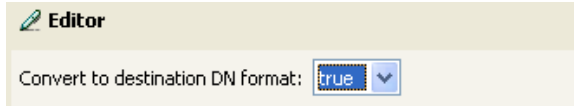
Conditions

- ✓ **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")))





2.7.24 XPath

Expands to results of evaluating an XPath 1.0 expression.

Fields

Expression

Specify the XPath 1.0 expression to evaluate.

Example

```
..... XPath("//*[ @attr-name='OU']/value[start-with(string(.),'xxx')]")
```

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 200](#)
- ◆ [Section 2.8.2, “Escape Source DN,” on page 201](#)
- ◆ [Section 2.8.3, “Lower Case,” on page 201](#)
- ◆ [Section 2.8.4, “Parse DN,” on page 202](#)
- ◆ [Section 2.8.5, “Replace All,” on page 204](#)
- ◆ [Section 2.8.6, “Replace First,” on page 205](#)
- ◆ [Section 2.8.7, “Substring,” on page 206](#)
- ◆ [Section 2.8.8, “Upper Case,” on page 207](#)

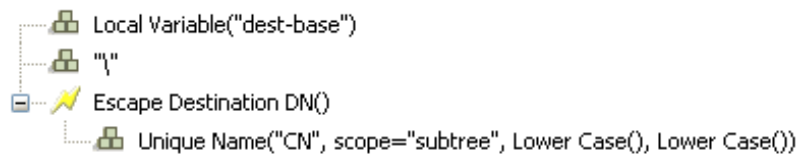
2.8.1 Escape Destination DN

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. For more information, see [“Placement - Publisher Flat” on page 98](#).

The action of Set Operation Destination DN uses the Escape Destination DN token to build the destination DN of the User object.

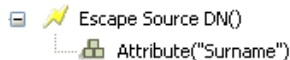


The Escape Destination DN token takes the value in Unique Name and sets it to the format for the destination DN.

2.8.2 Escape Source DN

Escapes a string according to the rules of the DN format of the source data store.

Example



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

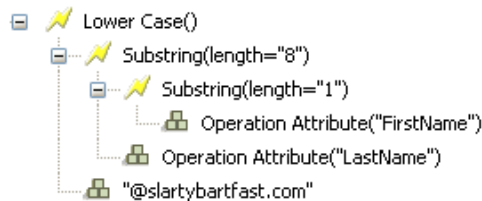
Conditions

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.

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.

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. For more

information, see “[Command Transformation - Create Departmental Container - Part 1 and Part 2](#)” on page 76.

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="organizational Unit", 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")))

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.

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.

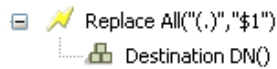
Remarks

For details on creating regular expressions, see:

- ♦ [Sun's Java Web site \(http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html\)](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



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(java.lang.String)) ([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. For more information, see [“Input or Output Transformation - Reformat Telephone Number from \(nnn\) nnn-nnnn to nnn-nnn-nnnn”](#) on page 88.

The Replace First token is used in the Reformat Operation Attribute action.

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

Conditions

Condition Group 1
Define new condition here

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")))

Replace First("^((\\d\\d\\d))s*(\\d\\d\\d)-(\\d\\d\\d\\d)\$", "\$1-\$2-\$3")
Local Variable("current-value")

Editor

Regular expression: * ^((\\d\\d\\d))s*(\\d\\d\\d)-(\\d\\d\\d\\d)\$

Replace with: \$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-xxx-xxxx.

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

For example, if the start is specified as -2, then it starts reading the first character from the end. If -3 is specified, then it starts 2 characters from the end.

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, -1 represents the entire length or the original string. If -2 is specified, the length is the entire -1. 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](#).

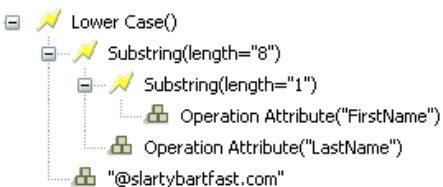
The screenshot displays the Identity Manager Policy Builder interface for a policy named "Set email address: name@slartybartfast.com; name = (1 char of Given Name + Surname) <= 8 chars". The interface is divided into two main sections: "Conditions" and "Actions".

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](#).

The screenshot shows the configuration for a policy named "Convert First/Last name to uppercase". 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 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")))

The diagram shows a lightning bolt icon representing the "Upper Case()" action, with a dashed line connecting it to a folder icon representing the "Operation Attribute("Given Name")" parameter.

2.9 Values

This section contains a list of common policy builder values.

2.9.1 Comparison Modes

Table 2-7 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 Sun's Java Web site (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html) and Sun's Java Web site (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#matches()) . 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.

Mode	Description
numeric	Compare numerically.
octet	Compare octet (Base64 encoded) values.
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

3

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 39](#)
- ◆ [Section 3.2, “Policy Builder Tasks in iManager,” on page 212](#)

This section also contains the following detailed reference sections:

- ◆ [Section 3.3, “Regular Expressions,” on page 244](#)
- ◆ [Section 3.4, “XPath 1.0 Expressions,” on page 245](#)
- ◆ [Section 3.5, “Conditions,” on page 246](#)
- ◆ [Section 3.6, “Actions,” on page 264](#)
- ◆ [Section 3.7, “Noun Tokens,” on page 305](#)
- ◆ [Section 3.8, “Verb Tokens,” on page 318](#)

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 246](#)) that must be met before a defined action (see [“Actions” on page 264](#)) 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 305](#)) and verbs (see [“Verb Tokens” on page 318](#)).
 - ◆ 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 244](#)) and XPath 1.0 expressions (see [“XPath 1.0 Expressions” on page 245](#)) 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.
- ◆ An operation is any element in the XDS document 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 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).
- ◆ An operation usually represents an event, a command, or a status.

- ♦ 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.
- ♦ A policy can also get additional context from outside of the document and cause side effects that are not reflected in the result document.

3.2 Policy Builder Tasks in iManager

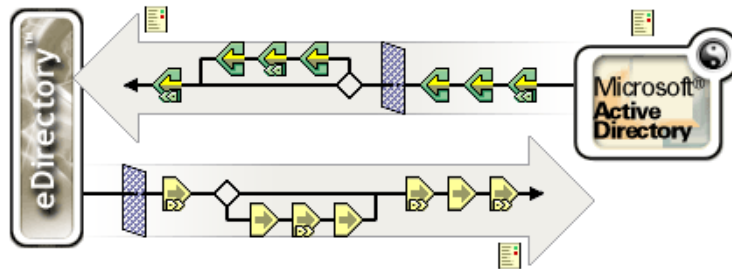
This section contains instructions on performing common tasks in the Policy Builder:

- ♦ [Section 3.2.1, “Opening The Policy Builder,” on page 212](#)
- ♦ [Section 3.2.2, “Creating a Policy,” on page 212](#)
- ♦ [Section 3.2.5, “Modifying a Policy,” on page 222](#)
- ♦ [Section 3.2.3, “Defining Individual Rules within a Policy,” on page 213](#)
- ♦ [Section 3.2.4, “Defining Individual Arguments within a Rule,” on page 214](#)
- ♦ [Section 3.2.12, “Using Predefined Rules,” on page 224](#)

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

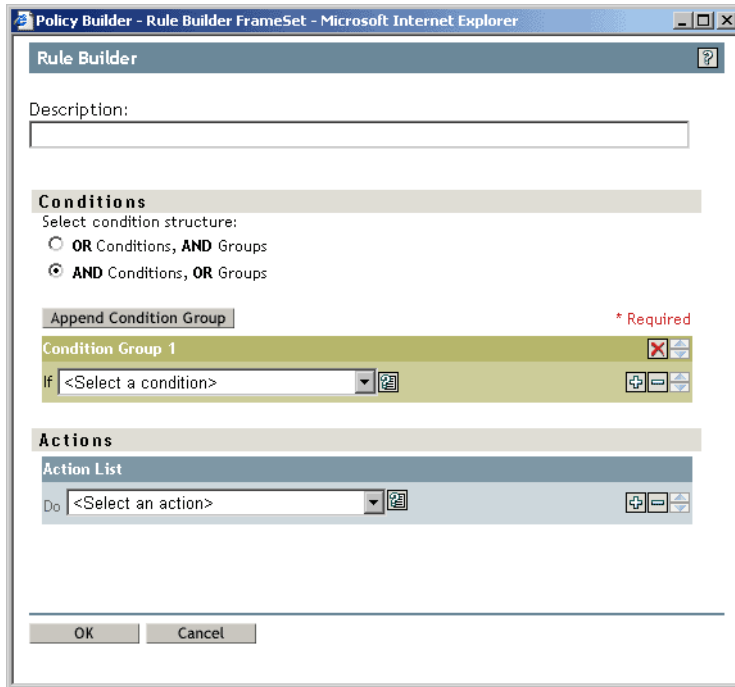
- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the icon representing the policy you want to define.
 - ◊ represents an undefined policy.
 - ➔ represents a defined policy.
- 3 Click *Insert*.
- 4 Enter a name for the new policy, then select the Policy Builder.

- 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 213.

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



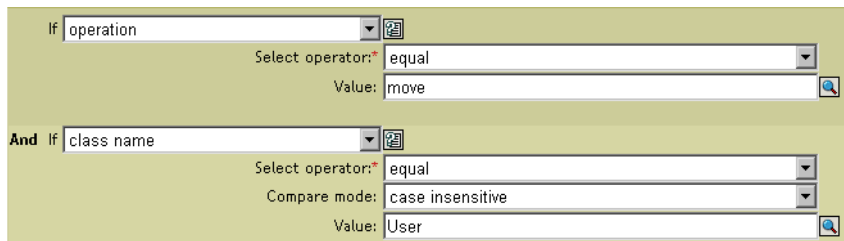
The Rule Builder interface enables you to quickly create and modify rules using intelligent drop-down menus.

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

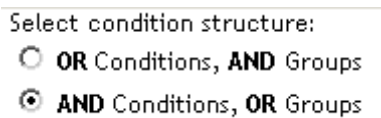













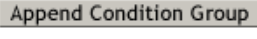


See [Section 3.5, “Conditions,” on page 246](#) and [Section 3.6, “Actions,” on page 264](#) or a detailed reference on the conditions and actions available in the Rule Builder.

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



- ◆ 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.

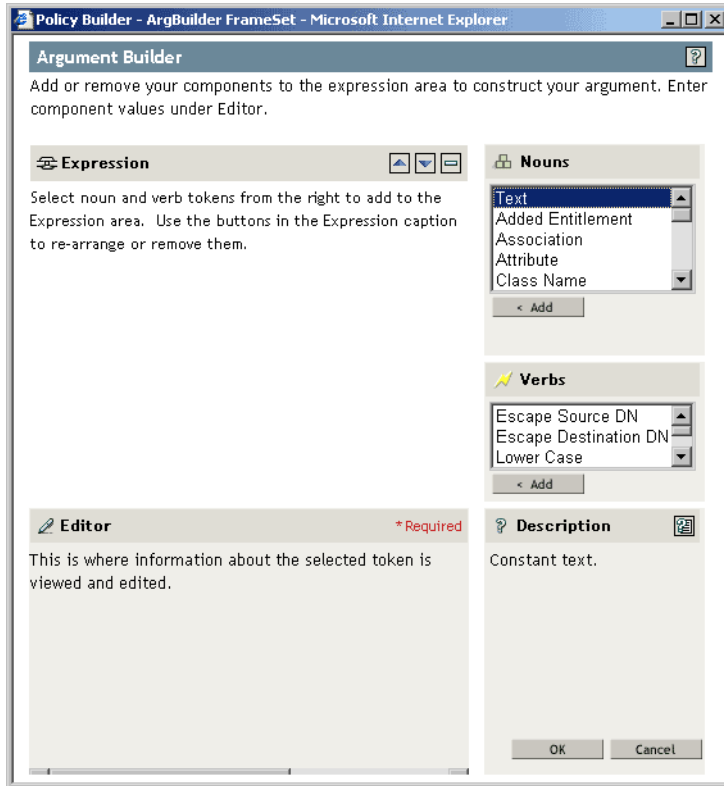
3.2.4 Defining Individual Arguments within a Rule

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 217](#).

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.

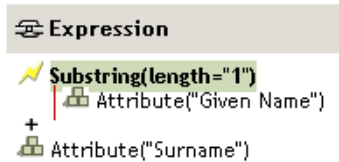
For example, if you want the argument set to an attribute value, you select the attribute token, then select the attribute name:

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 Give 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 305](#) and [Section 3.8, “Verb Tokens,” on page 318](#) 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:

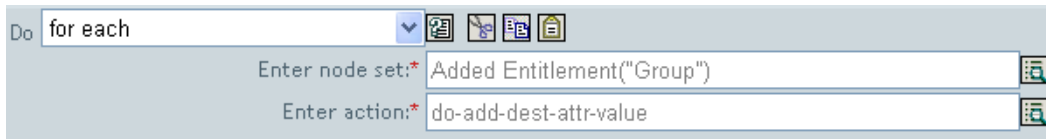
- ◆ [“Argument Actions Builder” on page 216](#)
- ◆ [“Argument Builder” on page 217](#)
- ◆ [“Match Attribute Builder” on page 218](#)
- ◆ [“Action Argument Component Builder” on page 219](#)
- ◆ [“Argument Value List Builder” on page 220](#)
- ◆ [“Named String Builder” on page 220](#)
- ◆ [“Condition Argument Component Builder” on page 221](#)

Argument Actions Builder

The Argument Actions Builder enables you to set the action that is required by the [For Each \(page 278\)](#) action and the [Implement Entitlement \(page 281\)](#) 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*

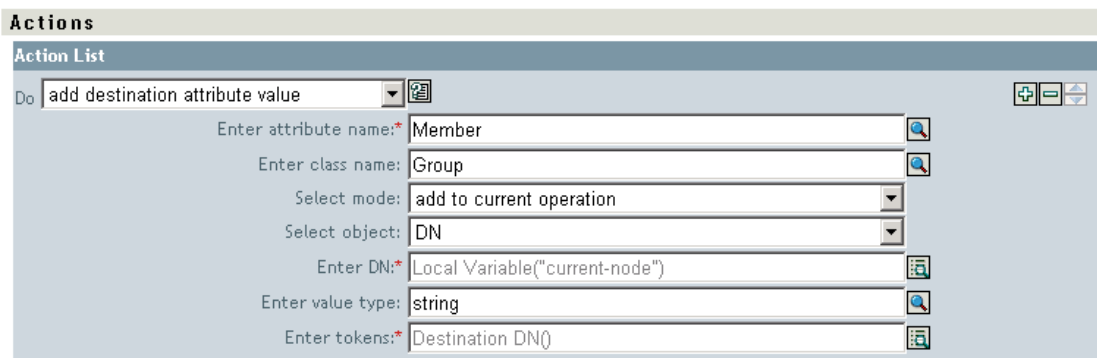
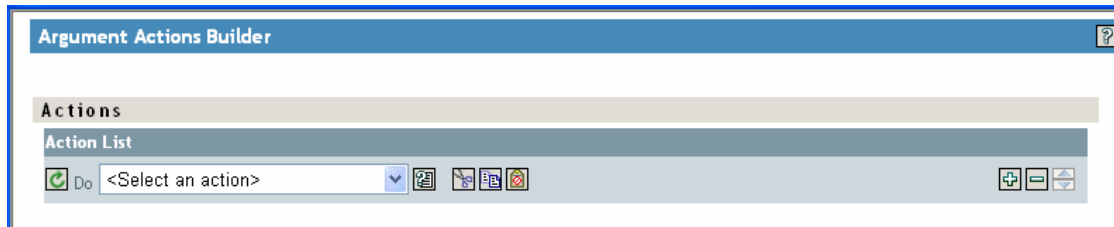


Figure 3-11 *Argument Actions Builder*



Argument Builder

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

- ◆ [Add Association \(page 265\)](#)
- ◆ [Add Destination Attribute Value \(page 266\)](#)
- ◆ [Add Destination Object \(page 267\)](#)
- ◆ [Add Source Attribute Value \(page 269\)](#)
- ◆ [Append XML Text \(page 271\)](#)
- ◆ [Clear Destination Attribute Value \(page 272\)](#) When the selected object is DN or Association.
- ◆ [Clear Source Attribute Value \(page 274\)](#) When the selected object is DN or Association.
- ◆ [Delete Destination Object \(page 276\)](#) When the selected object is DN or Association.

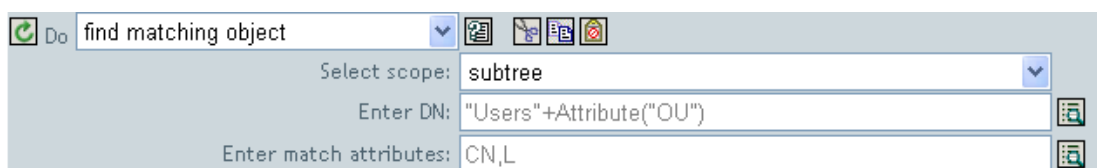
- ◆ [Delete Source Object \(page 276\)](#) When the selected object is DN or Association.
- ◆ [Find Matching Object \(page 276\)](#)
- ◆ [For Each \(page 278\)](#)
- ◆ [Move Destination Object \(page 282\)](#)
- ◆ [Move Source Object \(page 283\)](#)
- ◆ [Reformat Operation Attribute \(page 284\)](#)
- ◆ [Remove Association \(page 284\)](#)
- ◆ [Remove Destination Attribute Value \(page 285\)](#)
- ◆ [Remove Source Attribute Value \(page 286\)](#)
- ◆ [Rename Destination Object \(page 287\)](#) When the selected object is DN or Association and Enter String.
- ◆ [Rename Source Object \(page 288\)](#) When the selected object is DN or Association and Enter String.
- ◆ [Set Destination Attribute Value \(page 292\)](#) When the selected object is DN or Association and Enter Value type is not structured.
- ◆ [Set Destination Password \(page 293\)](#)
- ◆ [Set Local Variable \(page 294\)](#)
- ◆ [Set Operation Association \(page 295\)](#)
- ◆ [Set Operation Class Name \(page 295\)](#)
- ◆ [Set Operation Destination DN \(page 295\)](#)
- ◆ [Set Operation Property \(page 296\)](#)
- ◆ [Set Operation Source DN \(page 296\)](#)
- ◆ [Set Operation Template DN \(page 297\)](#)
- ◆ [Set Source Attribute Value \(page 297\)](#)
- ◆ [Set Source Password \(page 298\)](#)
- ◆ [Set XML Attribute \(page 300\)](#)
- ◆ [Status \(page 301\)](#)
- ◆ [Trace Message \(page 302\)](#)

Match Attribute Builder

The Match Attribute Builder enables you to select attributes and values used by the [Section 3.6.17, “Find Matching Object,” on page 276](#) 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

Match Attributes	
Name: * CN	Value from current object
Name: * L	Value from current object

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

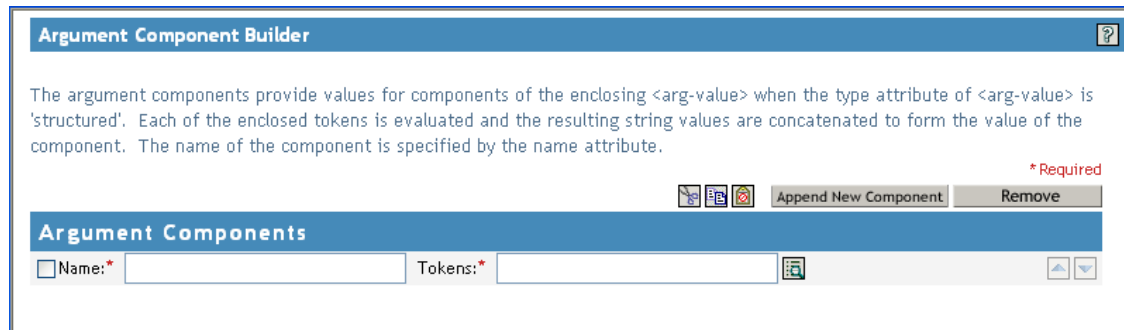
Launch the Action Argument Component Builder by selecting the following actions when the Enter Value Type selection is the Structured selection.

- ◆ [Add Destination Attribute Value \(page 266\)](#)
- ◆ [Add Source Attribute Value \(page 269\)](#)
- ◆ [Reformat Operation Attribute \(page 284\)](#)
- ◆ [Remove Destination Attribute Value \(page 285\)](#)
- ◆ [Remove Source Attribute Value \(page 286\)](#)
- ◆ [Set Default Attribute Value \(page 291\)](#)
- ◆ [Set Source Attribute Value \(page 297\)](#)

Figure 3-14 Action Argument Component Builder

Action List	
Do	add destination attribute value
Enter attribute name: *	Given Name
Enter class name:	User
Select mode:	write directly to destination datastore
Select object:	Current object
Enter value type:	structured
Enter components: *	user

Figure 3-15 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 \(page 291\)](#) 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 \(page 279\)](#), [Send Email \(page 288\)](#) and [Send Email from Template \(page 289\)](#).

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 \(page 247\)](#)
- ◆ [If Destination Attribute \(page 249\)](#)
- ◆ [If Source Attribute \(page 261\)](#)

Figure 3-20 *Structured Option*

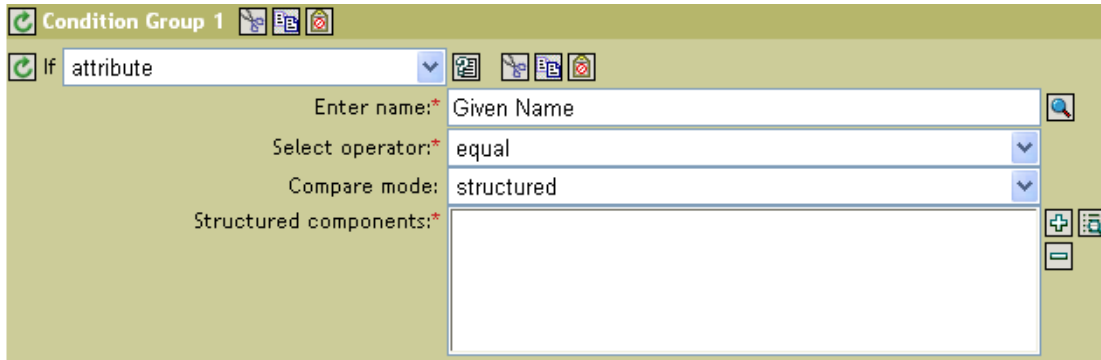
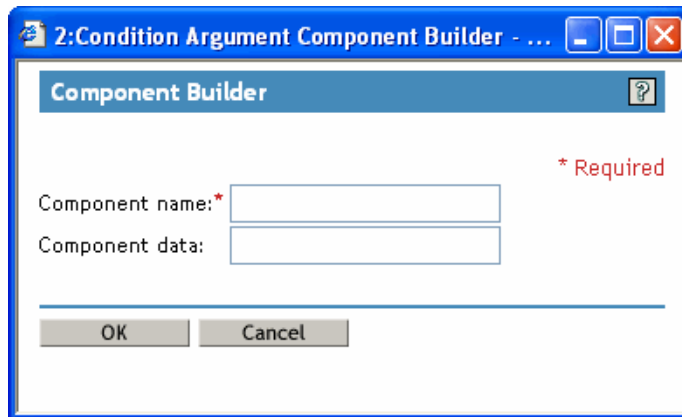


Figure 3-21 Condition Argument Component Builder



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[®] 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*.

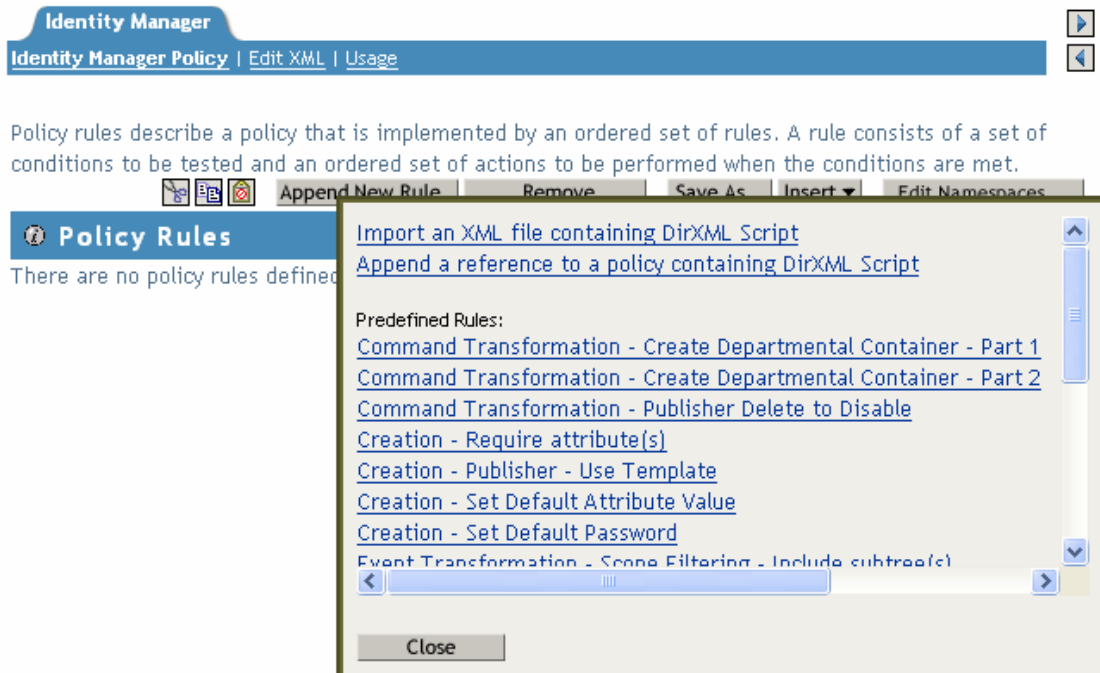
3.2.12 Using Predefined Rules

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 225
- ◆ “Command Transformation - Publisher Delete to Disable” on page 227
- ◆ “Creation - Require Attributes” on page 227
- ◆ “Creation - Publisher - Use Template” on page 228
- ◆ “Creation - Set Default Attribute Value” on page 229
- ◆ “Creation - Set Default Password” on page 230
- ◆ “Event Transformation - Scope Filtering - Include Subtrees” on page 231
- ◆ “Event Transformation - Scope Filtering - Exclude Subtrees” on page 232
- ◆ “Input or Output Transformation - Reformat Telephone Number from (nnn) nnn-nnnn to nnn-nnn-nnnn” on page 233
- ◆ “Input or Output Transformation - Reformat Telephone Number from nnn-nnn-nnnn to (nnn) nnn-nnnn” on page 234
- ◆ “Matching - Publisher Mirrored” on page 235
- ◆ “Matching - Subscriber Mirrored - LDAP Format” on page 236
- ◆ “Matching - By Attribute Value” on page 237
- ◆ “Placement - Publisher Mirrored” on page 238
- ◆ “Placement - Subscriber Mirrored - LDAP Format” on page 239
- ◆ “Placement - Publisher Flat” on page 240
- ◆ “Placement - Subscriber Flat - LDAP Format” on page 241
- ◆ “Placement - Publisher By Dept” on page 242
- ◆ “Placement - Subscriber By Dept - LDAP Format” on page 243

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 225](#).

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.

```
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")))
```

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

Editor

Name:*

Class name:

Select object:*

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 \(page 227\)](#).

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 228](#).

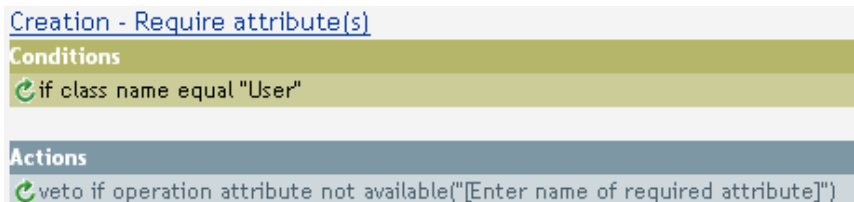
Creating a Policy

- 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 229](#).

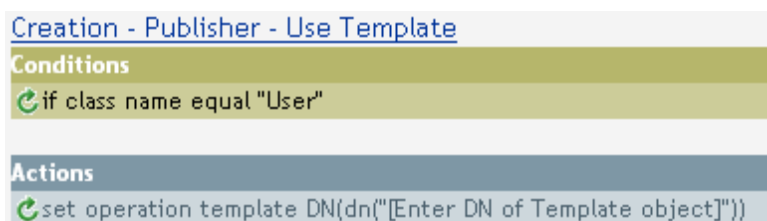
Creating a Policy

- 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 230](#).

Creating a Policy

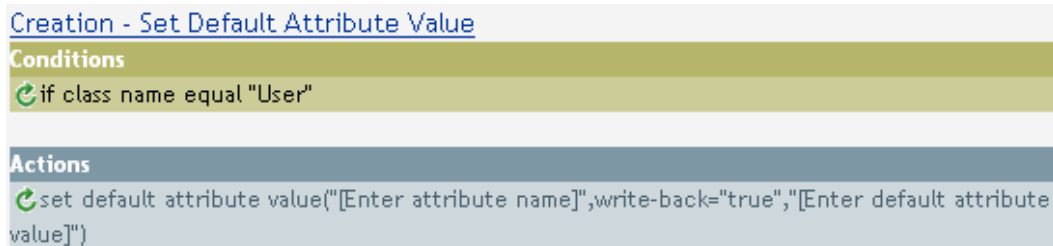
- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.

- 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 231](#).

Creating a Policy

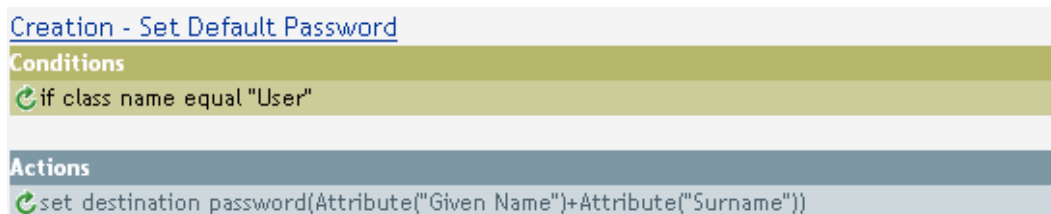
- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Creation Policy object on the Publisher or Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

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 232](#).

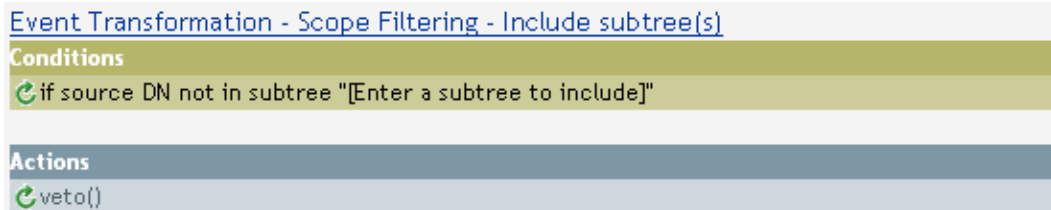
Creating a Policy

- 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 262](#) 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 232](#).

Creating a Policy

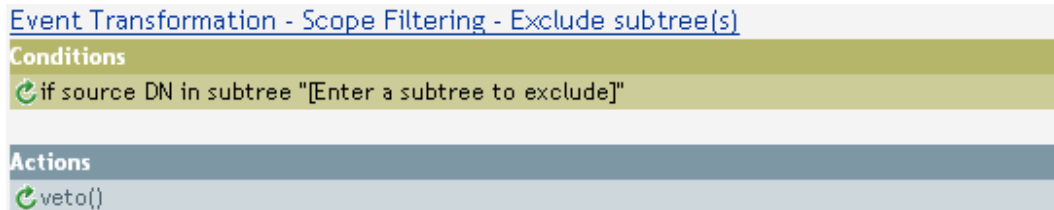
- 1 Open the Identity Manager Driver Overview for the driver you want to manage.
- 2 Click the Event Transformation Policy object on the Publisher or Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

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*.



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-nnnn

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 you then use the rule Reformat Telephone Number from nnn-xxx-nnnn 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 233](#).

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-nnnn*.
- 3 Click *Input or Output Transformation - Reformat Telephone Number from (nnn) nnn-nnnn to nnn-xxx-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 \(nnn\) nnn-nnnn to nnn-xxx-xxxx](#)

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-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” on page 234](#).

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-xxxx to (nnn) nnn-nnnn*.
- 3 Click *Input or Output Transformation - Reformat Telephone Number from nnn-xxx-xxxx 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 235](#).

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 object 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 236](#).

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 237](#).

Creating a Policy

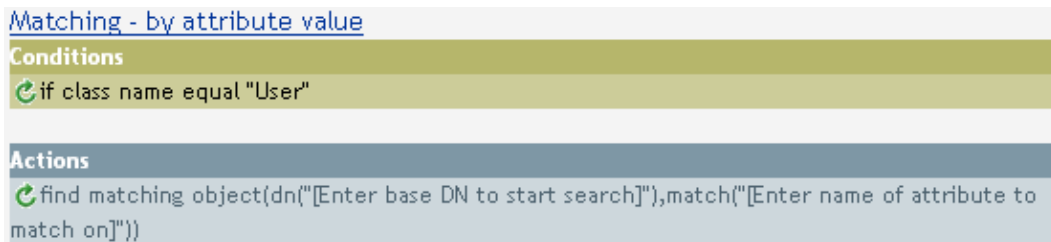
- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Matching Policy object on the Publisher channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

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*.



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 238](#).

Creating a Policy

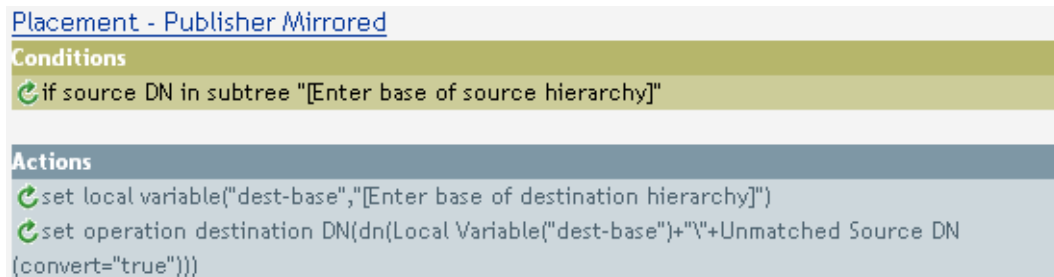
- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Placement Policy object on the Publisher channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

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*.



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 239](#).

Creating a Policy

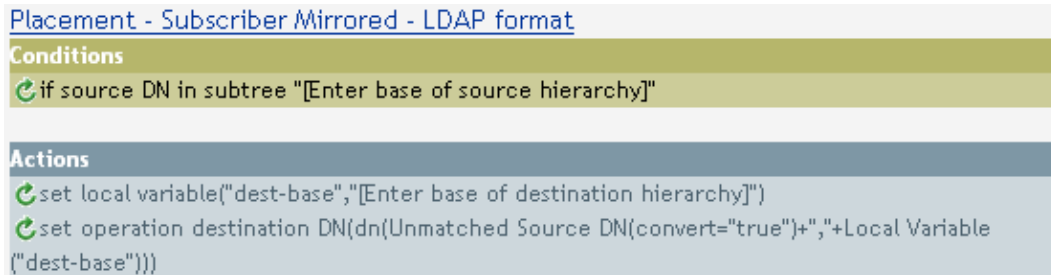
- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Placement Policy object on the Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

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*.



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 240](#).

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 241](#).

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 242](#).

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 243](#).

Creating a Policy

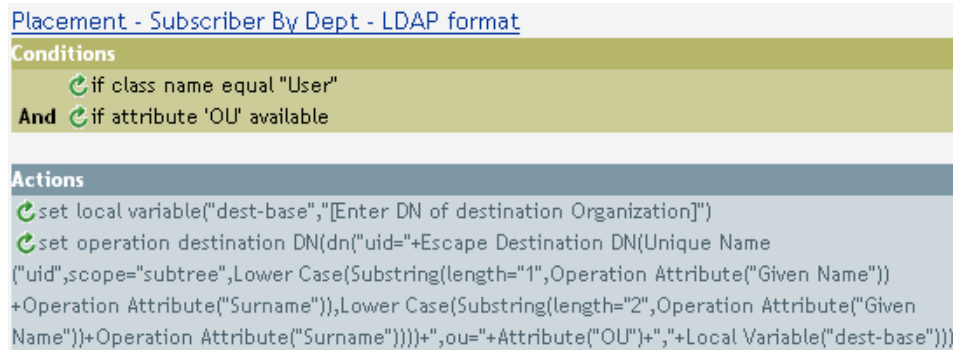
- 1 Open the *Identity Manager Driver Overview* for the driver you want to manage.
- 2 Click the Placement Policy object on the Subscriber channel.
- 3 Click *Insert*.
- 4 Name the policy, make sure to implement the policy with the Policy Builder, then click *OK*.

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*.



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.

Table 3-1 Common Regular Expressions

Metacharacter	Description
.	Matches any single character.
\$	Matches the end of the line.

Metacharacter	Description
^	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.
(?u)	Enables Unicode-aware case folding. This flag can impact performance.
(?i)	Enables case-insensitive matching.

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 246](#)
- ♦ [Section 3.5.2, “If Attribute,” on page 247](#)
- ♦ [Section 3.5.3, “If Class Name,” on page 248](#)
- ♦ [Section 3.5.4, “If Destination Attribute,” on page 249](#)
- ♦ [Section 3.5.5, “If Destination DN,” on page 250](#)
- ♦ [Section 3.5.6, “If Entitlement,” on page 251](#)
- ♦ [Section 3.5.7, “If Global Configuration Value,” on page 253](#)
- ♦ [Section 3.5.8, “If Local Variable,” on page 254](#)
- ♦ [Section 3.5.9, “If Named Password,” on page 256](#)
- ♦ [Section 3.5.10, “If Operation,” on page 256](#)
- ♦ [Section 3.5.11, “If Operation Attribute,” on page 258](#)
- ♦ [Section 3.5.12, “If Operation Property,” on page 259](#)
- ♦ [Section 3.5.13, “If Password,” on page 260](#)
- ♦ [Section 3.5.14, “If Source Attribute,” on page 261](#)
- ♦ [Section 3.5.15, “If Source DN,” on page 262](#)
- ♦ [Section 3.5.16, “If XPath Expression,” on page 263](#)

3.5.1 If Association

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

Fields

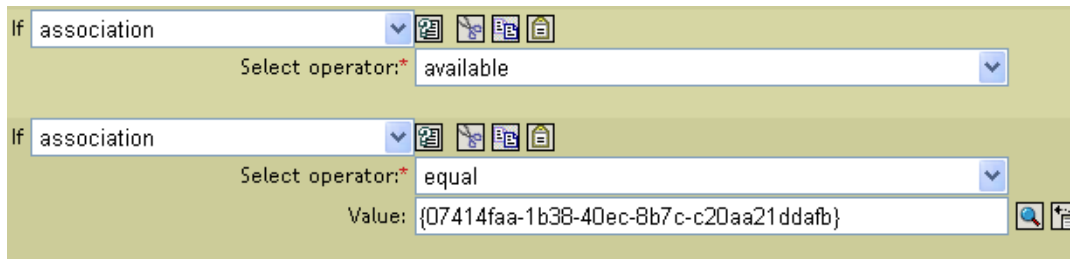
Operator Condition is Met When...

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.

Operator	Condition is met when...
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.



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

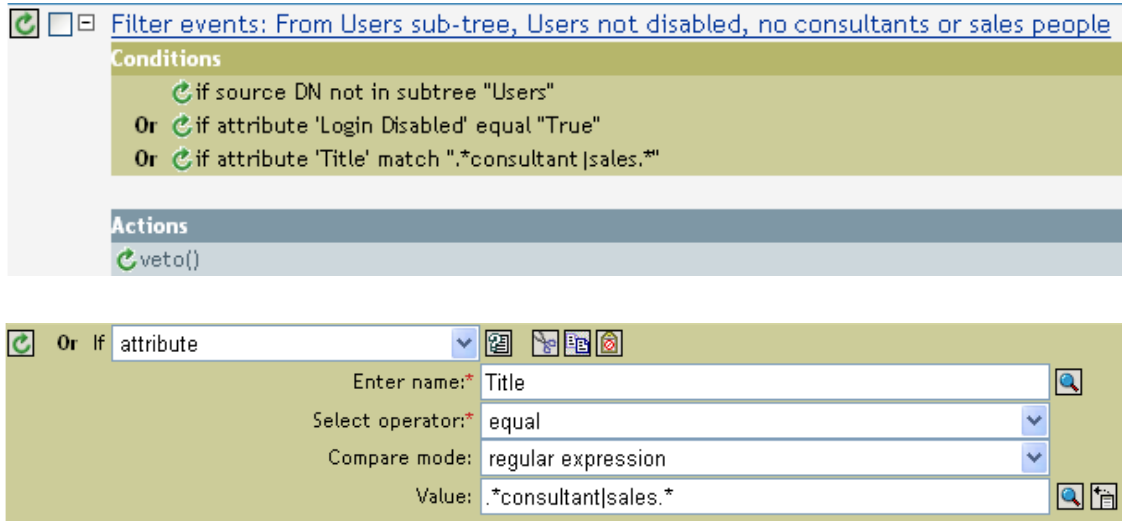
Select the comparison mode. See [“Comparison Modes” on page 326](#).

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 326](#).

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 the Policy Builder interface. It shows two policies: "User changing from Manager to Employee" and "User changing from Employee to Manager". The first policy is expanded to show its conditions and actions. The conditions are: "if class name equal 'User'", "And if destination attribute 'Title' match '.*manager.*'", and "And if operation attribute 'Title' not-match '.*manager.*'". The actions are: "set destination attribute value('Group Membership','Users\EmployeesGroup')" and "clone operation attribute('Group Membership','Security Equals')". Below the policies, a detailed view of the "If class name" condition is shown, with the following settings: "Select operator:" set to "equal", "Compare mode:" set to "case insensitive", and "Value:" set to "User".

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 326](#).

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.

Operator	Condition is met when...
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 the Policy Builder interface. At the top, two policies are listed:

- User changing from Manager to Employee**: Contains conditions:
 - if class name equal "User"
 - And if destination attribute 'Title' match ".*manager.*"
 - And if operation attribute 'Title' not-match ".*manager.*"
- User changing from Employee to Manager**: (Details are not fully visible)

Below the policies, a configuration dialog for the 'And If' condition is shown. The configuration is as follows:

- Condition type: **And If**
- Attribute: **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.
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 examples of operator conditions for destination DN in a Policy Builder interface. Each example consists of a text field for the destination DN, a dropdown menu for the operator, and a text field for the value (where applicable).

- Example 1:** If destination DN, Select operator: * available
- Example 2:** If destination DN, Select operator: * equal, Value: Novell\Users\Fred
- Example 3:** If destination DN, Select operator: * in container, Value: Novell\Users
- Example 4:** If destination DN, Select 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 326](#).

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.
not-equal	Equal would return False.

Example

The screenshot displays five stacked configuration panels for 'If entitlement' rules. Each panel has a dropdown menu set to 'entitlement' and a search icon. The configurations are as follows:

- Panel 1: Enter name: notes-group; Select operator: available
- Panel 2: Enter name: notes-group; Select operator: changing
- Panel 3: Enter name: notes-group; Select operator: changing from; Compare mode: case insensitive; Value: Sales
- Panel 4: Enter name: notes-group; Select operator: changing to; Compare mode: case insensitive; Value: Sales
- Panel 5: Enter name: notes-group; Select operator: equal; Compare mode: case insensitive; Value: Sales

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 326](#).

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 'If' conditions in a Policy Builder interface. The first condition is set to 'global configuration value' with the name 'myGlobalVariable' and the operator 'available'. The second condition is also set to 'global configuration value' with the name 'myGlobalVariable', the operator 'equal', the compare mode 'case insensitive', and the value '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 326](#).

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.

Operator	Condition is met when...
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

- 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.

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

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.

Value

The values are the operations that the Metadirectory engine looks for in this condition:

- ♦ add
- ♦ add-association
- ♦ check-object-password
- ♦ delete
- ♦ get-named-password
- ♦ modify
- ♦ modify-association
- ♦ modify-password
- ♦ move
- ♦ init-params
- ♦ instance

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](#).

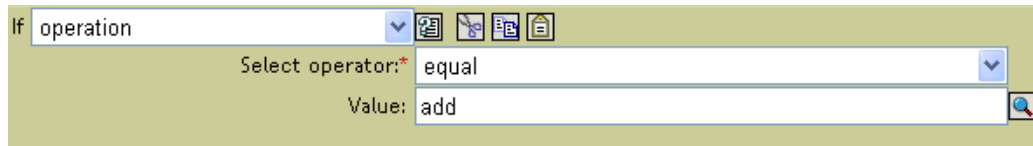
The screenshot shows a policy configuration window titled "Set local variables to test existence of groups and for placement". The window is divided into two main sections: "Conditions" and "Actions".

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"))))



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 326](#).

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 the Identity Manager Policy Builder interface. The main configuration area shows a policy with the following steps:

- 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

A detailed view of the 'And If' condition is shown below, with the following settings:

- 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 326](#).

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

The image shows two examples of configuring an 'If' condition. The top example has 'operation property' as the condition type, 'myStoredVariable' as the name, and 'available' as the operator. The bottom example has 'operation property' as the condition type, 'myStoredVariable' as the name, 'equal' as the operator, 'case insensitive' as the compare mode, and 'true' as the value.

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

The image shows an example of configuring an 'If' condition with 'password' as the condition type and 'available' as the operator.

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 326](#).

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 326](#).

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 'string(EN)' and 'string(JP)' as structured components. 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. For more information, see [“Event Transformation - Scope Filtering - Exclude Subtrees”](#) on page 232.

Event Transformation - Scope Filtering - Exclude subtree(s)

Conditions

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

Actions

veto()

If source DN

Select operator: in container

Value: Users

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

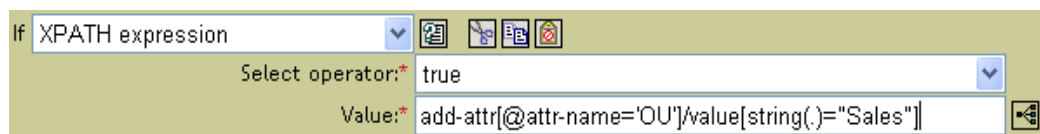
Operator

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 265](#)
- ◆ [Section 3.6.2, “Add Destination Attribute Value,” on page 266](#)
- ◆ [Section 3.6.3, “Add Destination Object,” on page 267](#)
- ◆ [Section 3.6.4, “Add Source Attribute Value,” on page 269](#)
- ◆ [Section 3.6.5, “Add Source Object,” on page 269](#)
- ◆ [Section 3.6.6, “Append XML Element,” on page 270](#)
- ◆ [Section 3.6.7, “Append XML Text,” on page 271](#)
- ◆ [Section 3.6.8, “Break,” on page 272](#)
- ◆ [Section 3.6.9, “Clear Destination Attribute Value,” on page 272](#)
- ◆ [Section 3.6.10, “Clear Operation Property,” on page 273](#)
- ◆ [Section 3.6.11, “Clear SSO Credential,” on page 273](#)
- ◆ [Section 3.6.12, “Clear Source Attribute Value,” on page 274](#)
- ◆ [Section 3.6.13, “Clone By XPath Expression,” on page 274](#)
- ◆ [Section 3.6.14, “Clone Operation Attribute,” on page 275](#)
- ◆ [Section 3.6.15, “Delete Destination Object,” on page 276](#)
- ◆ [Section 3.6.16, “Delete Source Object,” on page 276](#)
- ◆ [Section 3.6.17, “Find Matching Object,” on page 276](#)
- ◆ [Section 3.6.18, “For Each,” on page 278](#)
- ◆ [Section 3.6.19, “Generate Event,” on page 279](#)
- ◆ [Section 3.6.20, “Implement Entitlement,” on page 281](#)
- ◆ [Section 3.6.21, “Move Destination Object,” on page 282](#)

- ◆ Section 3.6.22, “Move Source Object,” on page 283
- ◆ Section 3.6.23, “Reformat Operation Attribute,” on page 284
- ◆ Section 3.6.24, “Remove Association,” on page 284
- ◆ Section 3.6.25, “Remove Destination Attribute Value,” on page 285
- ◆ Section 3.6.26, “Remove Source Attribute Value,” on page 286
- ◆ Section 3.6.27, “Rename Destination Object,” on page 287
- ◆ Section 3.6.28, “Rename Operation Attribute,” on page 287
- ◆ Section 3.6.29, “Rename Source Object,” on page 288
- ◆ Section 3.6.30, “Send Email,” on page 288
- ◆ Section 3.6.31, “Send Email from Template,” on page 289
- ◆ Section 3.6.32, “Set Default Attribute Value,” on page 291
- ◆ Section 3.6.33, “Set Destination Attribute Value,” on page 292
- ◆ Section 3.6.34, “Set Destination Password,” on page 293
- ◆ Section 3.6.35, “Set Local Variable,” on page 294
- ◆ Section 3.6.36, “Set Operation Association,” on page 295
- ◆ Section 3.6.37, “Set Operation Class Name,” on page 295
- ◆ Section 3.6.38, “Set Operation Destination DN,” on page 295
- ◆ Section 3.6.39, “Set Operation Property,” on page 296
- ◆ Section 3.6.40, “Set Operation Source DN,” on page 296
- ◆ Section 3.6.41, “Set Operation Template DN,” on page 297
- ◆ Section 3.6.42, “Set Source Attribute Value,” on page 297
- ◆ Section 3.6.43, “Set Source Password,” on page 298
- ◆ Section 3.6.44, “Set SSO Credential,” on page 299
- ◆ Section 3.6.45, “Set SSO Passphrase,” on page 299
- ◆ Section 3.6.46, “Set XML Attribute,” on page 300
- ◆ Section 3.6.47, “Status,” on page 301
- ◆ Section 3.6.48, “Strip Operation Attribute,” on page 301
- ◆ Section 3.6.49, “Strip XPath,” on page 302
- ◆ Section 3.6.50, “Trace Message,” on page 302
- ◆ Section 3.6.51, “Veto,” on page 303
- ◆ Section 3.6.52, “Veto if Operation Attribute Not Available,” on page 304

3.6.1 Add Association

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

Fields

Mode

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

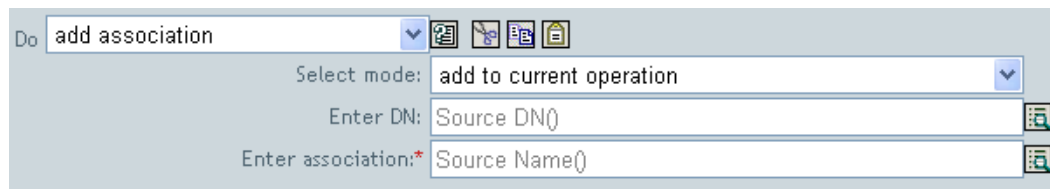
DN

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



The screenshot shows a software interface with a title bar containing the text "Do add association" and several icons. Below the title bar, there is a "Select mode:" dropdown menu with the value "add to current operation". Below that, there are two text input fields: "Enter DN:" with the value "Source DN()" and "Enter association:*" with the value "Source Name()". Both input fields have small icons to their right.

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. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2”](#) on page 225.

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 266](#) 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. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2” on page 225](#) 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 \(page 269\)](#) 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 image shows a screenshot of a software interface with two configuration panels. The first panel is titled "add source object" and contains the following fields: "Enter class name:*" with the value "User", and "Enter DN:*" with the value "Users/Fred Flintstone". The second panel is titled "add source attribute value" and contains the following fields: "Enter attribute name:*" with the value "Surname", "Enter class name:" (empty), "Select object:" with a dropdown menu showing "DN", "Enter DN:*" with the value "Users/Fred Flintstone", "Enter value type:" with the value "string", and "Enter string:*" with the value "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 configuration steps in a policy builder interface:

- Step 1:** Action: `append XML element`. Name: `jdbc:statement`. XPath expression: `..`
- Step 2:** Action: `append XML element`. Name: `jdbc:sql`. XPath expression: `../jdbc:statement[last()]`
- Step 3:** Action: `append XML text`. XPath expression: `../jdbc:statement[last()]/jdbc:sql`. String: `" UPDATE dirxml.emp SET fname = "+Operation Attribute`
- Step 4:** Action: `append XML text`. XPath expression: `../jdbc:statement[last()]/jdbc:sql`. 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 operations in a policy builder interface. Each operation is a 'Do' step with a dropdown menu set to 'append XML element' or 'append XML text'. The first two operations are 'append XML element' with names 'jdbc:statement' and 'jdbc:sql' respectively, and an XPath expression of '..'. The last two operations are 'append XML text' with an XPath expression of '..'/jdbc:statement[last()]/jdbc:sql' and a string value of '" 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' step in a policy builder interface with a dropdown menu set to '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.

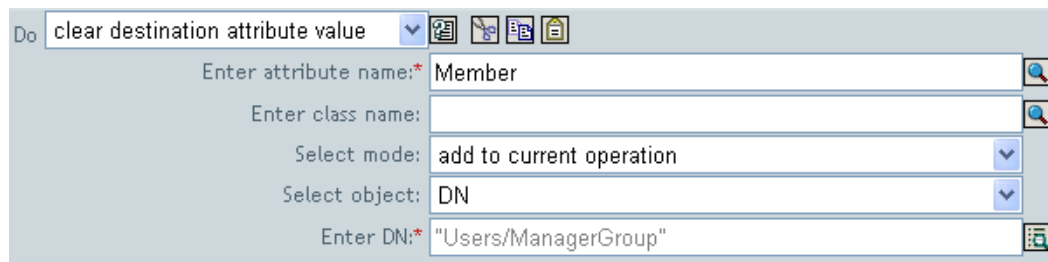
Mode

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



3.6.10 Clear Operation Property

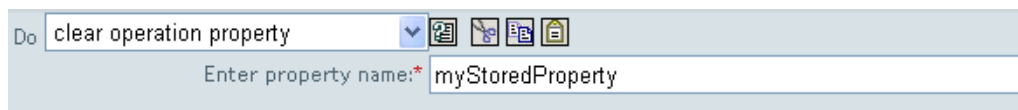
Clears any operation property current operation.

Fields

Property Name

Specify the name of the operation property to clear.

Example



3.6.11 Clear SSO Credential

Clears the Single Sign On credential, so objects can be deprovisioned. This action is part of the Credential Provisioning policies. For more information, see [Chapter 4, “Novell Credential Provisioning Policies,” on page 327.](#)

Fields

Credential Store Object DN

Specify the DN of the repository object.

Target User DN

Specify the DN of the target users.

Application Credential ID

Specify the application credential that is stored in the application object.

Login Parameter Strings

Specify each login parameter for the application. The login parameters are the authentication keys stored in the application object.

Example

The screenshot shows a dialog box titled "Do" with the action "clear SSO credential". It contains the following fields and options:

- Enter credential store object DN:*. \GroupWise_Repository
- Render browsed DN relative to policy
- Enter target user DN:*. Destination Attribute("DirXML-ADContext",class name="U: [Populate the following from an application object](#)
- Enter application credential ID:*. GroupWise_Credential
- Enter login parameter strings: Username,Password

3.6.12 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 dialog box titled "Do" with the action "clear source attribute value". It contains the following fields and options:

- Enter attribute name:*. Member
- Enter class name:
- Select object: DN
- Enter DN:*. "Users/ManagerGroup"

3.6.13 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 with a dropdown menu set to "clone by XPATH expressions". Below it are two input fields: "Enter source XPATH expression:*" with the value "@*" and "Enter destination XPATH expression:*" with the value "../modify[last()]".

3.6.14 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](#).

The screenshot displays a list of policy conditions and actions. The conditions include: "Set local variables to test existence of groups and for placement", "Create ManagersGroup, if needed", "Create EmployeesGroup, if needed", and "If Title indicates Manager, add to ManagerGroup and set rights". The actions include: "set destination attribute value('Group Membership', Local Variable('manager-group-dn'))" and "clone operation attribute('Group Membership', 'Security Equals')". Below the list is a configuration window for the "clone operation attribute" action, with "Group Membership" in the source name field and "Security Equals" in the destination name field.

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.

3.6.15 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"

3.6.16 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

Do delete source object

Select object: DN

Enter DN:* "Users/Fred Flintstone"

3.6.17 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 ￼. If multiple results are returned, then the destination DN of the current operation is set to the single character �.

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. For more information, see [“Matching - By Attribute Value” on page 94](#).

[Matching - by attribute value](#)

Conditions

if class name equal "User"

Actions

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

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.

Match Attributes

Name:* CN Value from current object

Name:* L Value from current object

3.6.18 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 281 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()

3.6.19 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.
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"> ◆ 0 = None ◆ 1 = Slash Notation ◆ 2 = Dot Notation ◆ 3 = LDAP Notation
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 list of four policy rules:

- [Setup Local Variables](#)
- [Surname A-I: place in Users1](#)
 - 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](#)
- [Surname S-Z: place in Users3](#)

Do generate event

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.

Strings

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 Trainging\Users\Active\Users1 container.

3.6.20 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

Action List

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 image shows a screenshot of a software interface titled "Action List". It contains a list of actions, with the first one selected and its configuration fields visible. The action is named "add destination attribute value". The configuration fields are: "Enter attribute name:" with the value "Login Disabled"; "Enter class name:" with the value "User"; "Select mode:" with the value "add to current operation"; "Select object:" with the value "DN"; "Enter DN:" with the value "Local Variable('current-node')"; "Enter value type:" with the value "string"; and "Enter string:" with the value "Destination DN()". There are several icons for editing and deleting actions.

3.6.21 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.

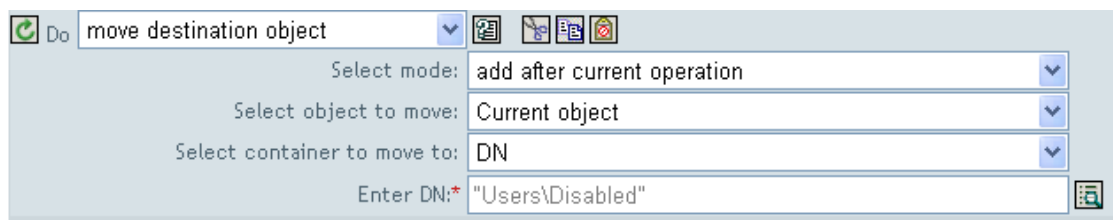
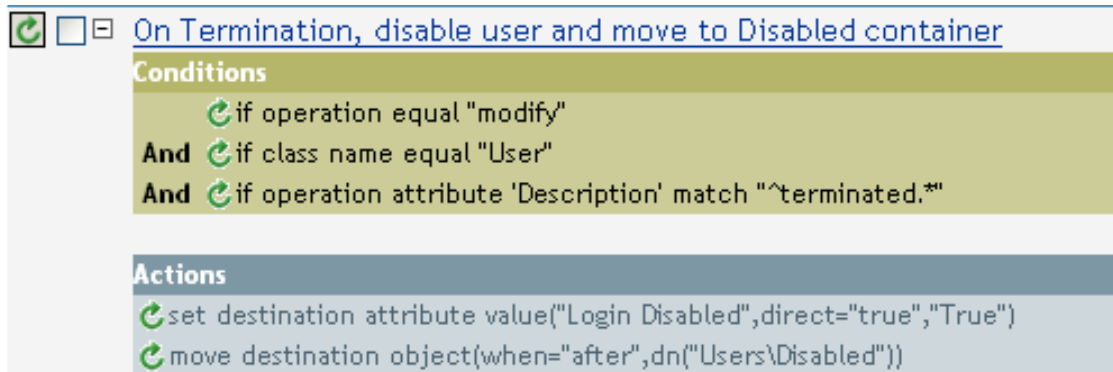
Container

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](#).



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.

3.6.22 Move Source Object

Moves an object in the source data store.

Fields

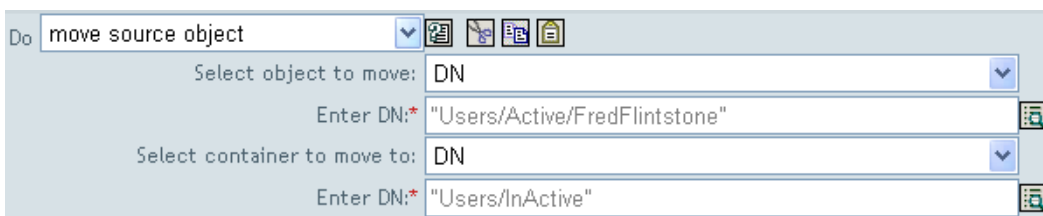
Object to Move

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



3.6.23 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 construct 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. For more information, see [“Input or Output Transformation - Reformat Telephone Number from \(nnn\) nnn-nnnn to nnn-nnn-nnnn”](#) on page 233.

[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.

3.6.24 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. For more information, see [“Command Transformation - Publisher Delete to Disable” on page 227](#).

The screenshot displays the configuration for a rule named "Command Transformation - Publisher Delete to Disable". It is divided into two main sections: "Conditions" and "Actions".

Conditions:

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

Actions:

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

Below the rule configuration, there is a control panel for the "remove association" action. It includes a dropdown menu for "Do" (set to "remove association"), a "Select mode:" dropdown (set to "add to current operation"), and an "Enter association:*" text field containing "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.

3.6.25 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 box with the following fields and values:

- Do: remove destination attribute value
- 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()

3.6.26 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.

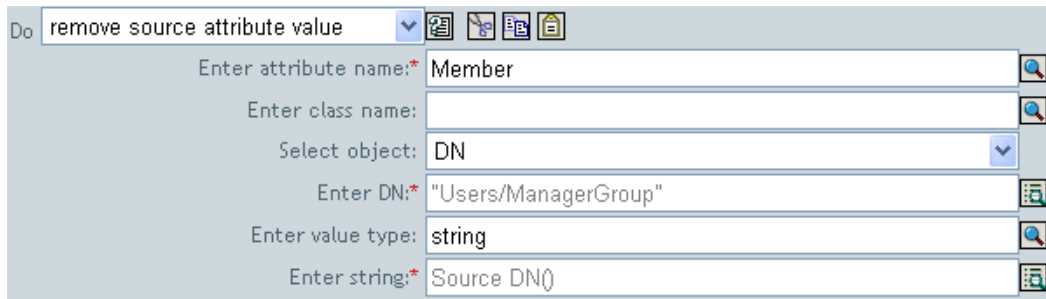
Value Type

Specify the syntax of the attribute value to be removed.

Value

Specify the attribute value to be removed.

Example



3.6.27 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.

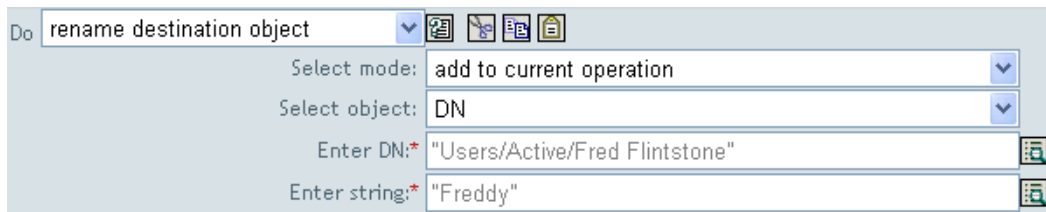
Object

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



3.6.28 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

Do rename operation attribute

Enter source name:* Surname

Enter destination name: sn

3.6.29 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

Do rename source object

Select object: DN

Enter DN:* "Users/Active/Fred Flintstone"

Enter string:* "Freddy"

3.6.30 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.
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"

3.6.31 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

The screenshot shows a configuration window titled "Do send email from template". It contains the following fields:

- Enter notification DN:*/cn=security/cn=Default Notification Collection
- Enter template DN:*/cn=security/cn=Default Notification Collection/cn=PS-Syr
- Enter password: (empty field)
- 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" 

3.6.32 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. For more information, see [“Creation - Set Default Attribute Value” on page 229](#).






[Creation - Set Default Attribute Value](#)


Conditions


 if class name equal "User"


Actions

 set default attribute value("[Enter attribute name]",write-back="true","[Enter default attribute value]")



 Do set default attribute value    

Enter attribute name:* company 

Write back: true 

Enter argument values:* "Digital Airlines" 

Argument Values

Type:* string  Enter string:* "Digital Airlines Inc" 

To build the value, the Argument Value List Builder is launched. See [“Argument Value List Builder” on page 220](#) 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.

3.6.33 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. For more information, see [“Command Transformation - Publisher Delete to Disable” on page 227](#).

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

Conditions:

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

Actions:

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

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 217](#) for more information about the builder.

3.6.34 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. For more information, see [“Creation - Set Default Password” on page 230](#).

[Creation - Set Default Password](#)

Conditions

- if class name equal "User"

Actions

- set destination password(Attribute("Given Name")+Attribute("Surname"))

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

3.6.35 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](#).

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

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"))))

A detailed view of the "set local variable" action is shown below, with the following fields:

- Do: set local variable
- Enter variable name: manager-group-info
- Select variable type: String
- Enter string: Destination Attribute("Object Class",dn(Local Variable("manager-group-dn")))

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 217](#) for more information.

3.6.36 Set Operation Association

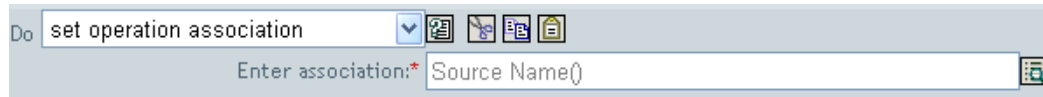
Sets the association value for the current operation.

Fields

Association

Provide the new association value.

Example



3.6.37 Set Operation Class Name

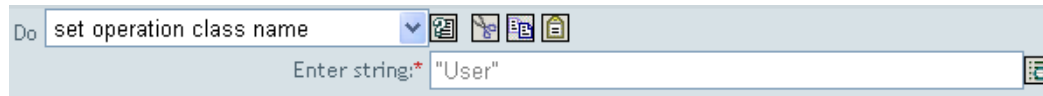
Sets the object class name for the current operation.

Fields

String

Specify the new class name.

Example



3.6.38 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. For more information, see [“Creation - Set Default Attribute Value” on page 82](#).

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

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

3.6.39 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:*

Enter string:*

3.6.40 Set Operation Source DN

Sets the source DN for the current operation.

Fields

DN

Specify the new source DN.

Example



3.6.41 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.

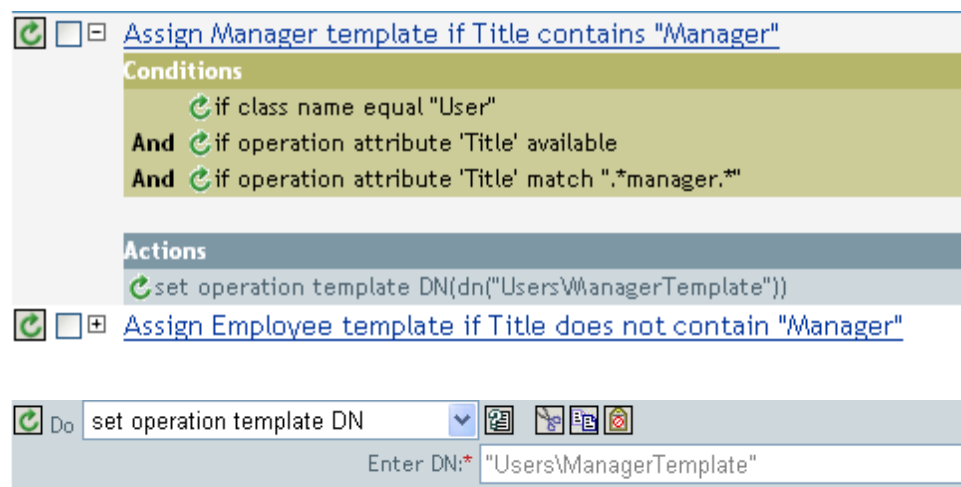
Fields

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.



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.

3.6.42 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

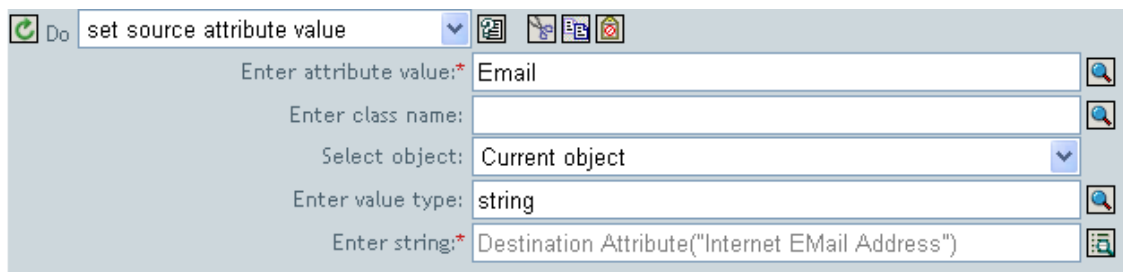
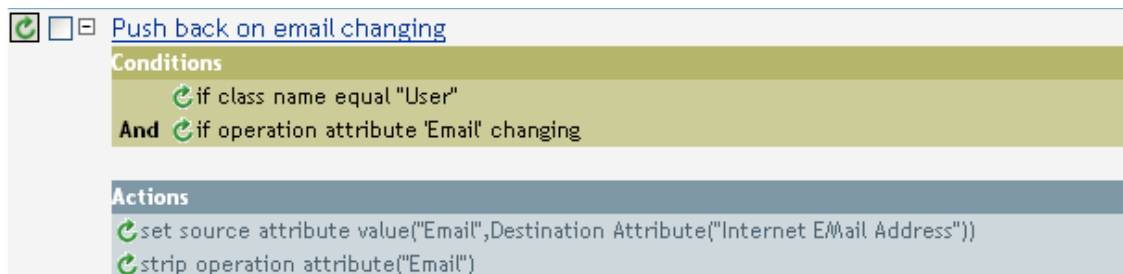
Select the syntax of the attribute value.

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](#).



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

3.6.43 Set Source Password

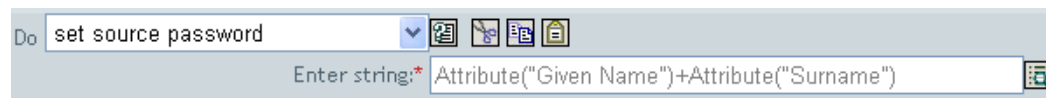
Sets the password for the current object in the source data store.

Fields

String

Specify the password to be set.

Example



The screenshot shows a dropdown menu with 'set source password' selected. Below it, a text input field contains the expression 'Attribute("Given Name")+Attribute("Surname")'. The interface includes standard icons for help, copy, paste, and refresh.

3.6.44 Set SSO Credential

Sets the SSO credential when a user object is created or when a password is modified. This action is part of the Credential Provisioning policies. For more information, see [Chapter 4, “Novell Credential Provisioning Policies,” on page 327.](#)

Fields

Credential Store Object DN

Specify the DN of the repository object.

Target User DN

Specify the DN of the target users.

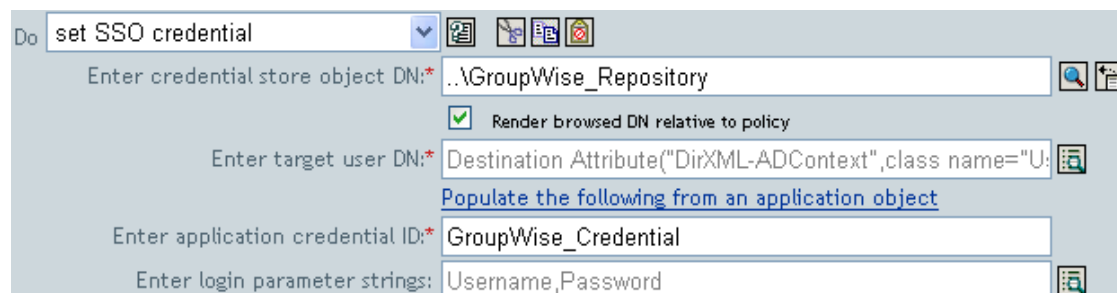
Application Credential ID

Specify the application credential that is stored in the application object.

Login Parameter Strings

Specify each login parameter for the application. The login parameters are the authentication keys stored in the application object.

Example



The screenshot shows a dropdown menu with 'set SSO credential' selected. Below it, there are four input fields: 'Enter credential store object DN:*' with the value '..\GroupWise_Repository', 'Enter target user DN:*' with the value 'Destination Attribute("DirXML-ADContext",class name="U:)', 'Enter application credential ID:*' with the value 'GroupWise_Credential', and 'Enter login parameter strings:' with the value 'Username,Password'. A checkbox labeled 'Render browsed DN relative to policy' is checked. A link 'Populate the following from an application object' is visible below the target user DN field.

3.6.45 Set SSO Passphrase

Sets the Novell SecureLogin[®] passphrase and answer when a User object is provisioned. This action is part of the Credential Provisioning policies. For more information, see [Chapter 4, “Novell Credential Provisioning Policies,” on page 327.](#)

Fields

Credential Store Object DN

Specify the DN of the repository object.

Target User DN

Specify the DN of the target users.


Question and Answer Strings

Specify the SecureLogin passphrase question and answer.

Example

The screenshot shows a configuration form for 'set SSO passphrase'. It includes the following fields and options:

- Do:** set SSO passphrase
- Enter credential store object DN:** ..\GroupWise_Repository
- Render browsed DN relative to policy**
- Enter target user DN:** Destination Attribute("DirXML-ADContext",class name="U:)
- Enter question and answer strings:** "Employee code?","Attribute("workforceID")"

The SecureLogin passphrase question and answer are stored as strings in the policy. Click the *Edit these strings* icon  to launch the string builder. Specify the passphrase question and answer.

Credential passphrase information is specified by two string elements. The first string contains the question and the second string contains the answer.

The screenshot shows the 'Strings' configuration window with the following fields:

- Question:** "Employee Code"
- Answer:** Destination Attribute("workforceID",class name="User")

** Required*

3.6.46 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 Policy Builder interface. Each instance has a dropdown menu set to 'set XML attribute' and 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 input field has a search icon on the right.

3.6.47 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 Policy Builder interface. The dropdown menu is set to 'status'. There are 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 input field has a search icon on the right.

3.6.48 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 titled "Push back on email changing". It is divided into two main sections: "Conditions" and "Actions".

- 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")

Below the main configuration area, there is a toolbar with a "Do" button and a dropdown menu set to "strip operation attribute". Below the toolbar is a text input field labeled "Enter name: *" with the value "Email" entered.

The action strips the attribute of Email. The value that is kept is what was in the destination Email attribute.

3.6.49 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 a policy configuration window titled "strip XPath expression". It features a toolbar with a "Do" button and a dropdown menu set to "strip XPath expression". Below the toolbar is a text input field labeled "Enter XPATH expression: *" with the value "*[@attr-name='OU']" entered.

3.6.50 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 3.0.1 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 screenshot displays the Identity Manager Policy Builder interface. It shows a list of rules with the following details:

- Setup Local Variables** (expanded)
- Surname A-I: place in Users1** (expanded)
 - 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** (expanded)
- Surname S-Z: place in Users3** (expanded)

Below the rule list, a detailed view of the 'trace message' action is shown:

- Do: trace message
- Enter level: []
- Select color: yellow
- Enter string: Local Variable("LVUsers1")

The action sends a trace message to DSTRACE. The contents of the local variable is LVUsers1 and it shows up in yellow in DSTRACE.

3.6.51 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. For more information, see [“Event Transformation - Scope Filtering - Exclude Subtrees” on page 232](#) from the predefined rules.

Event Transformation - Scope Filtering - Exclude subtree(s)

Conditions

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

Actions

- veto()

Do veto

The action vetoes all events that come from the specified subtree.

3.6.52 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](#).

User required attributes: First/Last Name, Title, Description, Email

Conditions

- if class name equal "User"

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")
- veto if operation attribute not available("Internet EMail Address")

Do veto if operation attribute not avail:

Enter name:* 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 305](#)
- ◆ [Section 3.7.2, “Association,” on page 306](#)
- ◆ [Section 3.7.3, “Attribute,” on page 306](#)
- ◆ [Section 3.7.4, “Class Name,” on page 307](#)
- ◆ [Section 3.7.5, “Destination Attribute,” on page 307](#)
- ◆ [Section 3.7.6, “Destination DN,” on page 308](#)
- ◆ [Section 3.7.7, “Destination Name,” on page 309](#)
- ◆ [Section 3.7.8, “Entitlement,” on page 309](#)
- ◆ [Section 3.7.9, “Global Configuration Value,” on page 310](#)
- ◆ [Section 3.7.10, “Local Variable,” on page 310](#)
- ◆ [Section 3.7.11, “Named Password,” on page 311](#)
- ◆ [Section 3.7.12, “Operation,” on page 311](#)
- ◆ [Section 3.7.13, “Operation Attribute,” on page 312](#)
- ◆ [Section 3.7.14, “Operation Property,” on page 313](#)
- ◆ [Section 3.7.15, “Password,” on page 313](#)
- ◆ [Section 3.7.16, “Removed Attribute,” on page 313](#)
- ◆ [Section 3.7.17, “Removed Entitlements,” on page 313](#)
- ◆ [Section 3.7.18, “Source Attribute,” on page 314](#)
- ◆ [Section 3.7.19, “Source DN,” on page 314](#)
- ◆ [Section 3.7.20, “Source Name,” on page 315](#)
- ◆ [Section 3.7.21, “Text,” on page 315](#)
- ◆ [Section 3.7.22, “Unique Name,” on page 316](#)
- ◆ [Section 3.7.23, “Unmatched Source DN,” on page 317](#)
- ◆ [Section 3.7.24, “XPath,” on page 318](#)

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. For more information on the predefined rule, see [“Command Transformation - Publisher Delete to Disable” on page 227](#).

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. For more information, see [“Creation - Set Default Password” on page 230](#).

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.

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: * 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 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. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2”](#) on page 225.

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

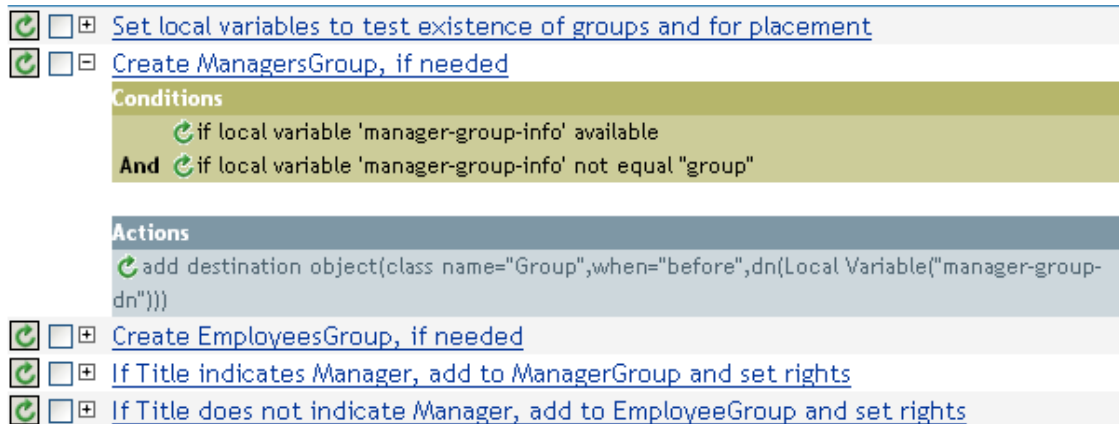
Name

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](#).

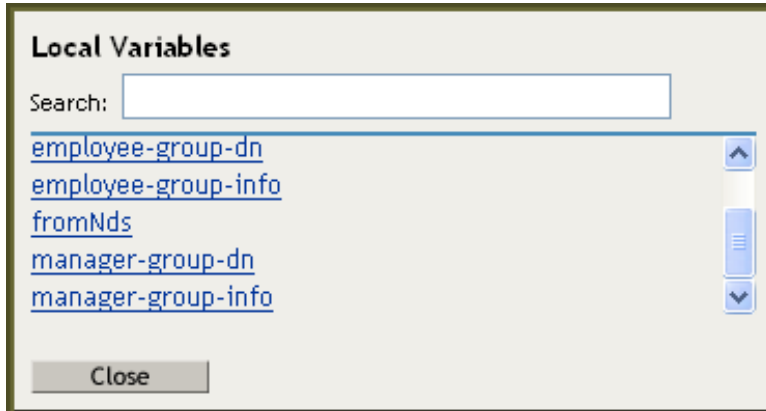
The action Add Destination Object uses the Local Variable token.



The screenshot shows a list of actions in a policy configuration tool. Each action has a green checkmark icon and a plus sign in a square. 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")



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 the specified attribute from the current XDS operation. It is different from Source Attribute and Destination Attribute, because it is always accessed directly from what is available in the current XDS operation as opposed to being queried from the source or destination data stores. 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](#).

The screenshot displays the configuration for a rule named "Surname A-I: place in Users1". The rule is expanded to show its conditions and actions.

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"))

Below the rule list, a variable selection pane shows the following variables:

- "Training\Users\Active\Users1"
- "\"
- Operation Attribute("CN")

At the bottom, an "Editor" window is open, showing a text input field for "Name: * CN" with a search icon to the right.

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

The XML attribute attached to an <operation-data> element by a policy. It is a place for policies to store and forward information for consumption by other policies.

Remarks

An XML attribute is a name value pair associated with an element in the XDS document.

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

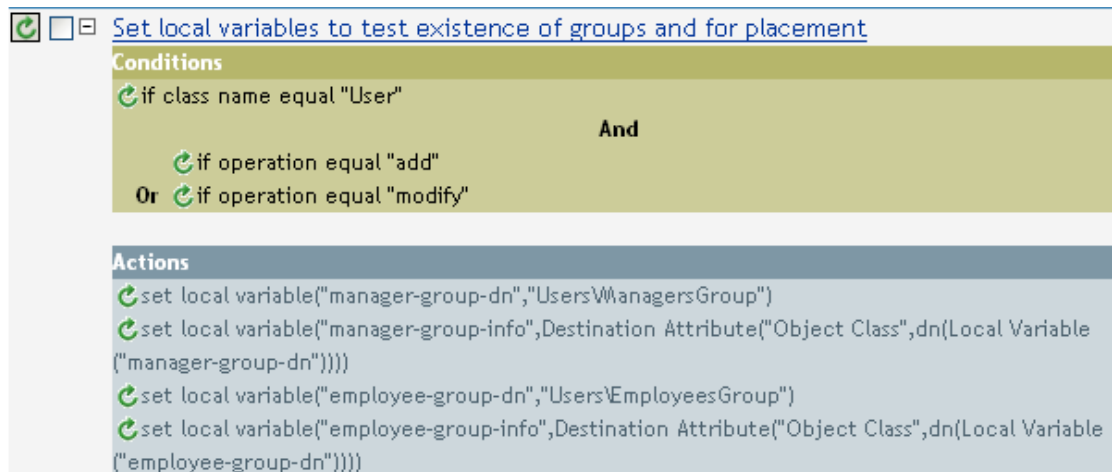
Text

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.




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

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"))))

 "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.

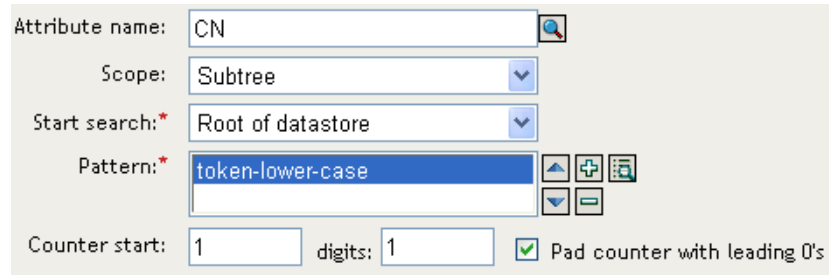
The counter can be set to start at a different number using the counter start field. The counter uses the number of digits specified by the digits field. If the number of digits is less than those specified, then the counter is right padded with zeros. When the number of digits exceeds those specified, 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: 

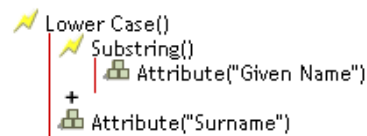
Scope: 

Start search: 

Pattern:   

Counter start: digits: Pad counter with leading 0's

The following pattern was constructed to provide unique names:



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. For more information, see [“Matching - Subscriber Mirrored - LDAP Format” on page 236](#).

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 319](#)
- ◆ [Section 3.8.2, “Escape Source DN,” on page 319](#)
- ◆ [Section 3.8.3, “Lower Case,” on page 319](#)
- ◆ [Section 3.8.4, “Parse DN,” on page 320](#)
- ◆ [Section 3.8.5, “Replace All,” on page 322](#)
- ◆ [Section 3.8.6, “Replace First,” on page 323](#)

- ◆ [Section 3.8.7, “Substring,” on page 324](#)
- ◆ [Section 3.8.8, “Upper Case,” on page 325](#)

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. For more information, see [“Placement - Publisher Flat” on page 98](#).

The action of Set Operation Destination DN uses the Escape Destination DN token to build the destination DN of the User object.

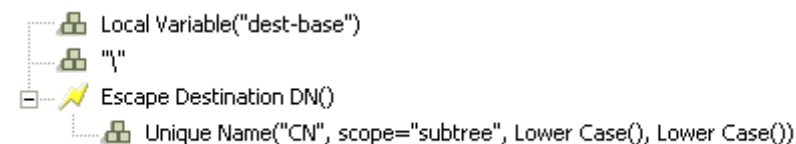
Placement - Publisher Flat

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

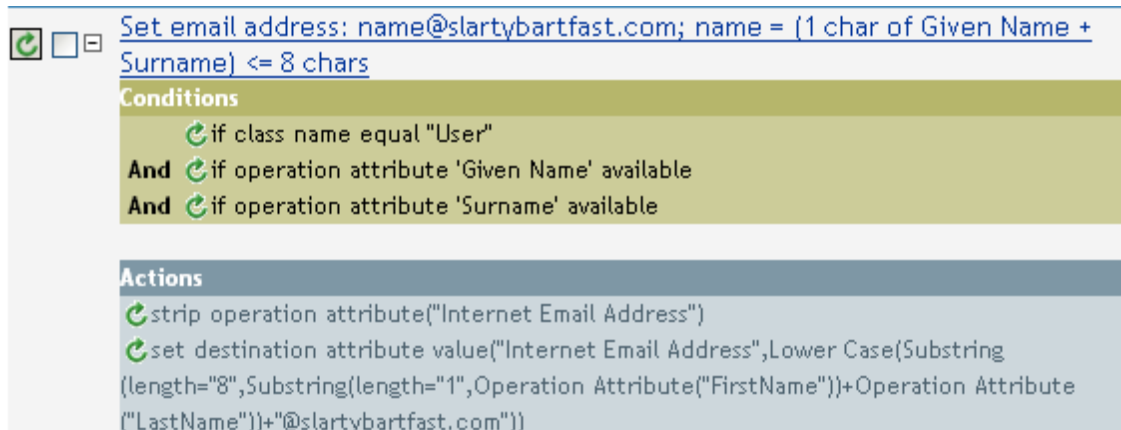
```
Escape Source DN()
  Attribute("Surname")
```

3.8.3 Lower Case

Converts the characters in a string to lowercase.

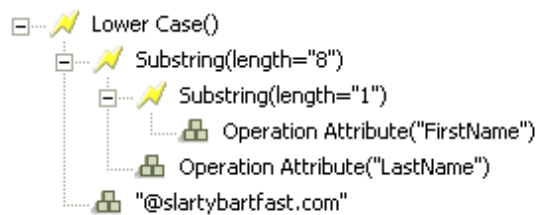
Example

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](#).



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

- Title:** 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. For more information, see [“Command Transformation - Create Departmental Container - Part 1 and Part 2”](#) on page 225.

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")))

Parse DN("dest-dn", "dot", length="1", start="-1")
Local Variable("target-container")

Editor

Start:	<input type="text" value="-1"/>
Length:	<input type="text" value="1"/>
Source DN format:	<input type="text" value="destination DN"/>
Destination DN format:	<input type="text" value="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.

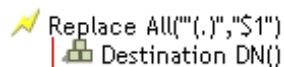
Remarks

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 ([java.lang.String](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 \(java.lang.String\)](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. For more information, see [“Input or Output Transformation - Reformat Telephone Number from \(nnn\) nnn-nnnn to nnn-nnn-nnnn” on page 233.](#)

The Replace First token is used in the Reformat Operation Attribute action.

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

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")))

☐ ⚡ Replace First("^((\d\d\d))s*(\d\d\d)-(\d\d\d\d)\$", "\$1-\$2-\$3")

 📦 Local Variable("current-value")

Editor

Regular expression: *

Replace with:

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-xxx-xxxx.

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.

For example, if the start is specified as -2, then it starts reading the first character from the end. If -3 is specified, then it starts 2 characters from the end.

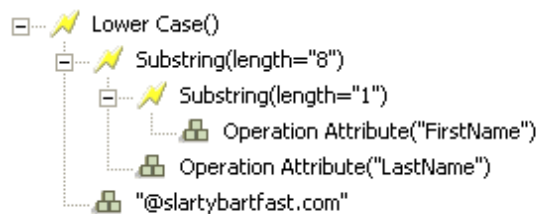
Length

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, -1 represents the entire length or the original string. If -2 is specified, the length is the entire -1. 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](#).

The screenshot shows the Policy Builder interface for a policy named "Push back on email changing". The policy is expanded to show its conditions and actions. The conditions section includes "if class name equal 'User'" and "And if operation attribute 'Email' changing". The actions section includes "set source attribute value('Email', Destination Attribute('Internet EMail Address'))" and "strip operation attribute('Email')".



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.

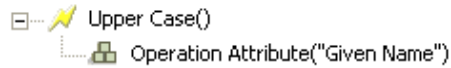
3.8.8 Upper Case

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](#).

The screenshot shows the Policy Builder interface for a policy named "Convert First/Last name to uppercase". The policy is expanded to show its conditions and actions. The conditions section includes "if class name equal 'User'", "And if operation attribute 'Given Name' changing", and "Or if operation attribute 'Surname' changing". The actions section includes "reformat operation attribute('Given Name', Upper Case(Operation Attribute('Given Name')))" and "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 Sun's Java Web site (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Pattern.html) and Sun's Java Web site (http://java.sun.com/j2se/1.4/docs/api/java/util/regex/Matcher.html#matches()).</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.

Novell Credential Provisioning Policies

4

Novell® Credential Provisioning Policies for Identity Manager 3.0.1 enhance the user provisioning abilities of any Identity Manager driver by providing the capability to simultaneously provision application credentials to the Novell SecretStore® and Novell SecureLogin credential repositories. Additionally, it can provision the SecureLogin passphrase question and answer in environments where non-repudiation is desired.

These features enhance the user Single Sign-On experience and increase the return on investment of Single Sign-On technologies by eliminating the initial setup of SecureLogin account information, providing additional security to application credentials, and reducing the replication of effort normally associated with provisioning Single Sign-On credential stores for users. In addition, the Credential Provisioning policies can use Identity Manager policies to automatically de-provision application credentials to prevent access to application data.

- ♦ [Section 4.1, “Credential Provisioning Policies with Novell SecureLogin,” on page 327](#)
- ♦ [Section 4.2, “Implementing Credential Provisioning Policies with Novell SecureLogin,” on page 329](#)
- ♦ [Section 4.3, “Credential Provisioning Policies with Novell SecretStore,” on page 349](#)
- ♦ [Section 4.4, “Implementing Credential Provisioning Policies with SecretStore,” on page 351](#)

4.1 Credential Provisioning Policies with Novell SecureLogin

Credential Provisioning policies allow you automatically provision application credentials that SecureLogin supports. This topic documents the steps required to configure objects and policies in Identity Manager. It does not contain deployment and configuration information for any SecureLogin components. For SecureLogin documentation, see [Novell SecureLogin 6.0 documentation \(http://www.novell.com/documentation/securelogin60/index.html\)](http://www.novell.com/documentation/securelogin60/index.html).

To implement Credential Provisioning with SecureLogin requires a repository object, an application object, and policies. The repository and application objects store the SecureLogin information so that Identity Manager can use it. The policies are used to enable a driver to use Credential Provisioning. See [Section 4.2, “Implementing Credential Provisioning Policies with Novell SecureLogin,” on page 329](#) for more information.

You can also configure the following options:

- ♦ Credential Provisioning can be provided by the Publisher channel, Subscriber channel, or both channels.
- ♦ SecureLogin synchronization can occur as part of an application password synchronization or can be triggered by some other event.
- ♦ Web Services credentials can be provisioned without provisioning accounts for the application.
- ♦ An initial SecureLogin passphrase question and answer can be provisioned.

Figure 4-1 on page 328 shows a typical, yet simple, scenario involving the provisioning of the SecureLogin credentials for a new User of a SAP* Finance application in a Finance department. SAP User provisioning is being utilized because it is an application that requires more login parameters than a typical username and password provided for most application.

This department provisions new users into the Identity Vault via a SAP HR system and Identity Manager. Depending on organizational information, the User object is then provisioned into a department authentication tree implemented on Active Directory. This is where new users authenticate to the network and is therefore the location for the SecureLogin credential repository. As users are subsequently provisioned by Identity Manager to the various finance applications, their credentials for those systems are synchronized to the SecureLogin store in Active Directory.

Figure 4-1 shows user Glen’s authentication credentials being provisioned. When Glen authenticates to his department’s Active Directory authentication domain and launches SecureLogin client, he has single sign-on to his SAP Finance account without ever needing to enter, or even know, his password on that system.

Figure 4-1 Credential Provisioning with SecureLogin

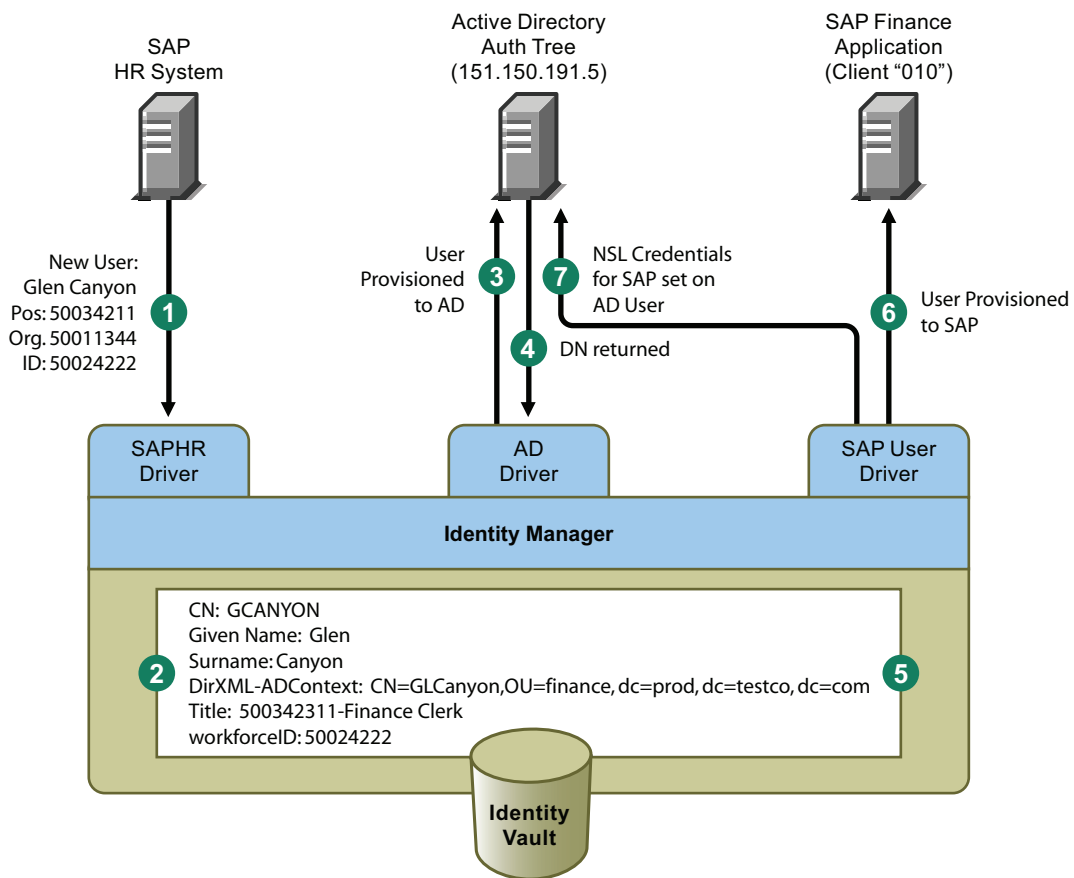


Figure 4-1 illustrates the following steps:

1. A SAP HR system publishes the data for a newly hired user named Glen Canyon. The Identity Manager SAP HR driver processes this data.
2. A new User object is created in the Identity Vault with a CN value of GCANYON and a workforceID value of 50024222. Because this user is assigned to the Finance organization of

his company, he needs to authenticate to the Finance department Active Directory server in the finance.prod.testco.com domain. The Identity Manager Active Directory driver that synchronizes that domain now uses the Identity Vault information.

3. Glen is provisioned to the Finance department Active Directory server.
4. The driver is configured to obtain Glen's fully distinguished LDAP name:
CN=GLCanyon,OU=finace,dc=prod,dc=testco,dc=com.
5. The driver places the name into the DirXML-ADContext attribute of the GCANYON user in the Identity Vault.

Now that the required attributes are available in the Identity Vault, the SAP User Management driver processes the attributes of the GCANYON object.

6. Because Glen is in the Finance organization, the driver provisions a SAP user account GCANYON on the SAP Finance server.
7. After the account creation is successful, the SAP User Management driver policies provision Glen's SAP authentication credentials to his AD user account. Because the command is an Add operation, the policies also provision his SecureLogin passphrase question and answer.

4.2 Implementing Credential Provisioning Policies with Novell SecureLogin

The implementation of Credential Provisioning policies with SecureLogin is very customizable. The steps to implement it are different depending upon the platforms SecureLogin is installed on, the applications that are provisioned, and which Identity Manager drivers are involved.

To implement Credential Provisioning policies with SecureLogin, see the following topics:

- ◆ [Section 4.2.1, "Meeting Requirements for Credential Provisioning Policies with Novell SecureLogin," on page 329](#)
- ◆ [Section 4.2.2, "Extending LDAP Schema for Novell SecureLogin," on page 330](#)
- ◆ [Section 4.2.3, "Determining Deployment Configuration Parameters for Novell SecureLogin," on page 330](#)
- ◆ [Section 4.2.4, "Creating a Repository Object for Novell SecureLogin," on page 333](#)
- ◆ [Section 4.2.5, "Creating an Application Object for Novell SecureLogin," on page 339](#)
- ◆ [Section 4.2.6, "Configuring Credential Provisioning Policies for Novell SecureLogin," on page 345](#)

4.2.1 Meeting Requirements for Credential Provisioning Policies with Novell SecureLogin

In order to use Credential Provisioning Policies with SecureLogin, the following must be in place:

- ◆ Identity Manager 3.0.1
- ◆ Supported on eDirectory™ 8.7x and eDirectory 8.8.1 or above; eDirectory 8.8 is not supported.
- ◆ Verify that `jssso.jar`, `idmcp.jar`, and `jnet.jar` are in the standard location for Identity Manager Java libraries.
- ◆ Novell SecureLogin 6.0 or above

After you have verified that your environment meets the requirements, proceed to [Section 4.2.2, “Extending LDAP Schema for Novell SecureLogin,”](#) on page 330.

4.2.2 Extending LDAP Schema for Novell SecureLogin

When SecureLogin is deployed on eDirectory servers, a tool called `ndsschema.exe` is utilized to extend the eDirectory schema with a set of SecureLogin attributes that are used to store encrypted credentials, policies, etc. on Users and container objects. These attributes are:

- ◆ Prot:SSO Auth
- ◆ Prot:SSO Entry
- ◆ Prot:SSO Entry Checksum
- ◆ Prot:SSO Profile
- ◆ Prot:SSO Security Prefs
- ◆ Prot:SSO Security Prefs Checksum

These attributes are specific to eDirectory and are required in order for the SecureLogin product to function. The provisioning API provided in Identity Manager 3.0 Support Pack 1 utilizes the LDAP namespace to perform its functions so that it can work with any SecureLogin credential store. In order to provide LDAP mappings to the attributes listed above, a second tool provided with the SecureLogin product must be utilized. The tool name is `ldapschema.exe`, and it is used in eDirectory environments to provide the LDAP namespace mapping to the eDirectory attributes.

After running `ldapschema.exe`, verify the mappings by checking the LDAP Group attribute map in iManager.

- 1 In iManager, click *LDAP > LDAP Options*.
- 2 Select the LDAP Group associated with your eDirectory servers that host SecureLogin.
- 3 From the LDAP Group properties page, select the *Attribute Map* option and verify the attributes above are mapped to the following *Primary LDAP Attributes*:
 - ◆ protocom-SSO-Auth-Data
 - ◆ protocom-SSO-Entries
 - ◆ protocom-SSO-Entries-Checksum
 - ◆ protocom-SSO-Profile
 - ◆ protocom-SSO-Security-Prefs
 - ◆ protocom-SSO-Security-Prefs-Checksum

After the schema is extended, proceed to [Section 4.2.3, “Determining Deployment Configuration Parameters for Novell SecureLogin,”](#) on page 330.

4.2.3 Determining Deployment Configuration Parameters for Novell SecureLogin

In order to provide the synchronization functionality described in the deployment scenario illustrated in [Figure 4-1](#), the first step is to gather all of the business process information related to the Identity Manager and SecureLogin environments. You can print [Table 4-1, “Credential](#)

Provisioning Policies Worksheet for SecureLogin,” on page 331, and use it as a worksheet to record the information.

Table 4-1 *Credential Provisioning Policies Worksheet for SecureLogin*

Configuration Information Needed	Information
1) Which applications will be configured for SecureLogin Single Sign-On provisioning?	
2) Verify that SecureLogin application definitions are preconfigured on the authentication server and are inheritable by new users provisioned to those systems.	
3) The DNS name or IP address of the SecureLogin repository server.	
4) The SSL LDAP port for the SecureLogin repository server.	
5) The fully qualified LDAP distinguished name of the administrator for the SecureLogin repository server.	
6) The password of the administrator for the SecureLogin repository server.	
7) The full path and the name of the SSL certificate exported from the SecureLogin server. The certificate must be local to the Identity Manager server.	
8) Determine if one SecureLogin repository will be used by multiple drivers or if each driver will use a separate repository.	
9) The application ID for each SecureLogin application.	
10) Find all required authentication keys for each application. Such as, Username, Password, Client, and Language. They might be different for each application.	
11) Determine if any of the authentication key values can be set with a static value.	
12) For non-static values that are or can be different for each user, make a note of the source of the non-static information (event information or Identity Vault attribute values).	
13) If you are implementing SecureLogin provisioning on a driver that is also synchronizing a password to the target application, determine if the SecureLogin provisioning takes place before or after the password is set in the target application server.	

Configuration Information Needed	Information
14) The name of the Driver object where the repository and application objects are to be stored. (Can be different drivers.)	
15) Determine the DN of the User objects for the target application.	
16) If you are implementing a SecureLogin passphrase, determine the passphrase question and answer.	Question: Answer:

Example Provisioning Configuration Data

Using the provisioning scenario, the following example data provisions a user's SecureLogin credentials for the SAP Finance server for users in the Finance Active Directory authentication tree:

Table 4-2 Example Credential Provisioning Policies Worksheet for SecureLogin

Configuration Information Needed	Information
1) Which applications will be configured for SecureLogin Single Sign-On provisioning?	SAP Finance Application
2) Verify that SecureLogin application definitions are preconfigured on the authentication server and are inheritable by new users provisioned to those systems.	Verified
3) The DNS name or IP address of the SecureLogin repository server.	151.150.191.5
4) The SSL LDAP port for the SecureLogin repository server.	636
5) The fully qualified LDAP distinguished name of the administrator for the SecureLogin repository server.	cn=admin,ou=prod,dc=testco,dc=.com
6) The password of the administrator for the SecureLogin repository server.	dixml
7) The full path and the name of the SSL certificate exported from the SecureLogin server. The certificate must be local to the Identity Manager server.	c:\novell\nds\FinanceAD.cer
8) Determine if one SecureLogin repository will be used by multiple drivers or if each driver will use a separate repository.	For this example, there is only one repository.
9) The application ID for each SecureLogin application.	SAP - 151.150.191.27
10) Find all required authentication keys for each application. Such as, Username, Password, Client, and Language. They might be different for each application.	SAP Client 010 Login Parameter Client SAP Client 010 Login Parameter Language SAP Client 010 Login Parameter Username SAP Client 010 Login Parameter Password

Configuration Information Needed	Information
11) Determine if any of the authentication key values can be set with a static value.	SAP Client 010 Login Parameter Client:"010" SAP Client 010 Login Parameter Language: "EN"
12) For non-static values that are or can be different for each user, make a note of the source of the non-static information (event information or Identity Vault attribute values).	SAP Client 010 Login Parameter Username: Identity Vault attribute "sapUsername" SAP Client 010 Login Parameter Password: Event <password>
13) If you are implementing SecureLogin provisioning on a driver that is also synchronizing a password to the target application, determine if the SecureLogin provisioning takes place before or after the password is set in the target application server.	After
14) The name of the Driver object where the repository and application objects are to be stored. (Can be different drivers.)	SAP driver
15) Determine the DN of the User objects for the target application.	Identity Vault attribute "DirXML-ADContext"
16) If you are going to provision the SecureLogin passphrase, determine the passphrase question and answer.	Question: "Employee code?" Answer: Identity Vault attribute "workforceID"

Miscellaneous Environment Information:

- ◆ The Finance department AD tree serves as the SecureLogin repository for all Finance applications.
- ◆ All finance department provisioning drivers are in a driver set called Finance Drivers.
- ◆ The SAP user account must be deleted and the SecureLogin credentials for the SAP user account must be removed from the Active Directory user when the Identity Vault attribute "employeeStatus" is set to the value "I".

After all of the configuration data has been determined, proceed to [Section 4.2.4, "Creating a Repository Object for Novell SecureLogin," on page 333](#).

4.2.4 Creating a Repository Object for Novell SecureLogin

Repository objects store static configuration information for SecureLogin. Repository information is independent from the applications that consume the application credentials. This information is applicable for all provisioning events regardless of the connected system (for example SAP, PeopleSoft*, Notes*, etc.). The repository object can be created in Designer or iManager.

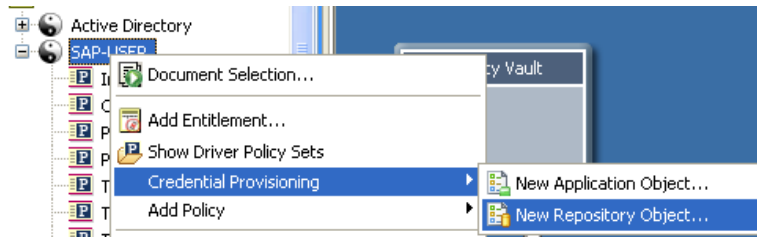
- ◆ ["Creating a Repository Object for Novell SecureLogin in Designer" on page 333](#)
- ◆ ["Creating a Repository Object for Novell SecureLogin in iManager" on page 337](#)

Creating a Repository Object for Novell SecureLogin in Designer

The following is one of many methods you can use to create the repository object in Designer.

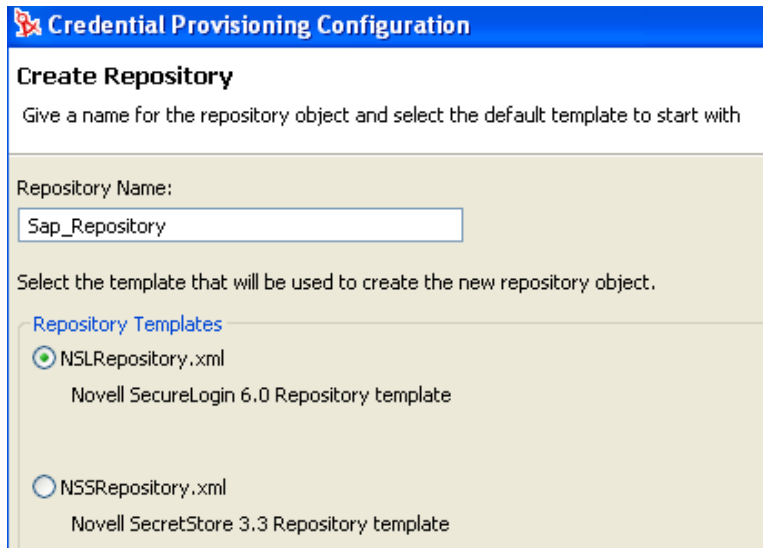
- 1 Right-click the driver object where you want to store the repository object in the outline view.

2 Click *Credential Provisioning* > *New Repository Object*.



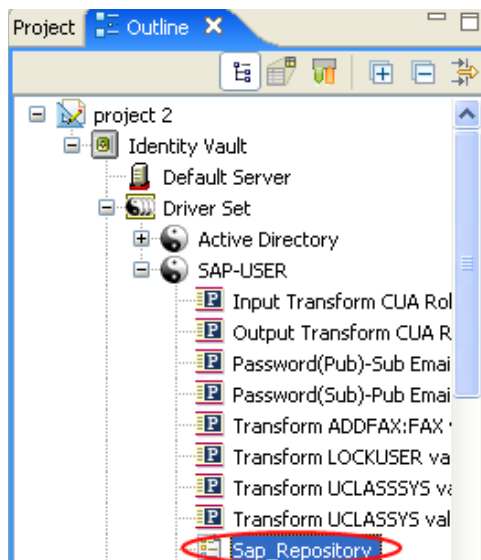
3 Specify a name for the repository object.

4 Select *NSLRepository.xml* to use the SecureLogin template.

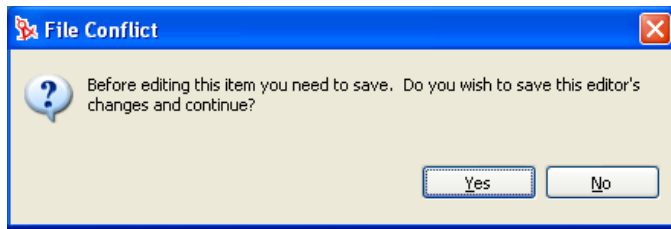


5 Click *OK*.

6 Double-click the repository object in the outline view to add configuration information.




7 Click *Yes* to save the new repository object.



8 Specify the DNS name or IP address of the SecureLogin server. See worksheet item 3).

SecureLogin Server Name or Address:

9 Specify the SSL port for the SecureLogin server. See worksheet item 4).

SecureLogin Server SSL Port: 

10 Specify the full path to the SSL certificate exported from the SecureLogin server. The path must include the certificate name and must be local to the Identity Manager server. See worksheet item 7).


SecureLogin Server SSL Certificate Path:

The SecureLogin server can run on multiple platform types. Refer to the platform-specific documentation for information on how to export the SSL certificates.

11 Specify the fully qualified LDAP distinguished name of the SecureLogin administrator. See worksheet item 5).

SecureLogin Administrator:



12 Click *Set password*.

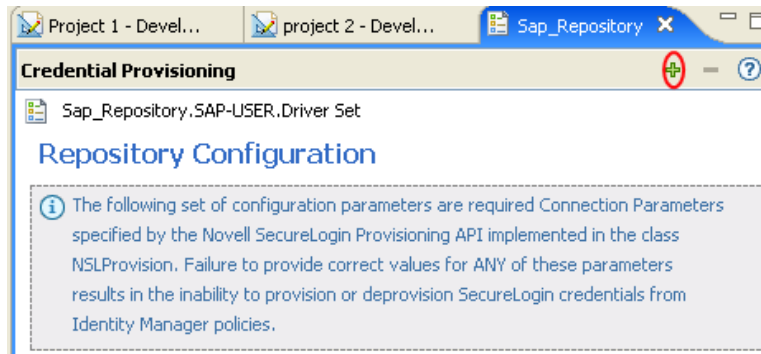
SecureLogin Administrator Password: 

- 13 Specify the SecureLogin administrator's password twice, then click *OK*. See worksheet item 6).



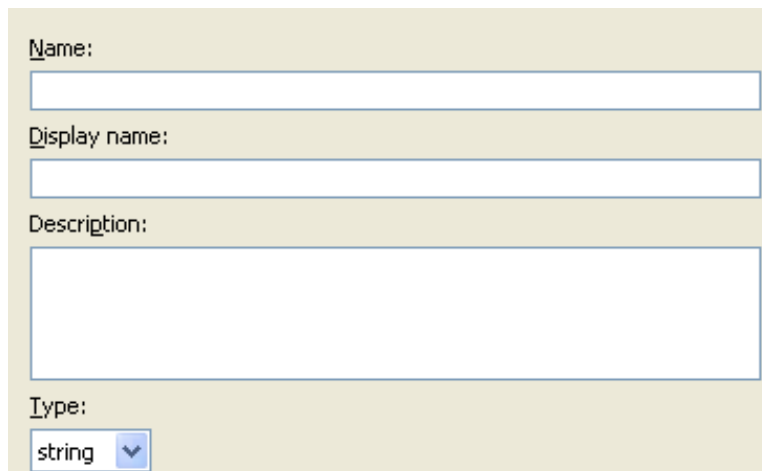
A dialog box titled "Change Password" with a close button (X) in the top right corner. It contains two text input fields. The first field is labeled "Enter password:" and contains six dots. The second field is labeled "Re-enter password:" and contains six dots and a cursor. At the bottom, there are two buttons: "OK" and "Cancel".

- 14 Review the information, then click the *Save* icon  to save the information.
- 15 (Optional) If you want to create other configuration parameters for the repository object, click the *Add new item*  icon.



A screenshot of a web-based console window titled "Credential Provisioning". The breadcrumb path is "Sap_Repository.SAP-USER.Driver Set". The main heading is "Repository Configuration". Below the heading is an information icon (i) and a text box containing the following text: "The following set of configuration parameters are required Connection Parameters specified by the Novell SecureLogin Provisioning API implemented in the class NSLProvision. Failure to provide correct values for ANY of these parameters results in the inability to provision or deprovision SecureLogin credentials from Identity Manager policies." In the top right corner of the console window, there is a red circle highlighting a plus sign icon.

- 15a Specify a name for the parameter.
- 15b Specify a display name for the parameter.
- 15c Specify a description for the parameter for your reference.
The parameter is stored as a string.



A form for configuring a parameter. It has four sections: "Name:" with a text input field; "Display name:" with a text input field; "Description:" with a large text area; and "Type:" with a dropdown menu currently set to "string".

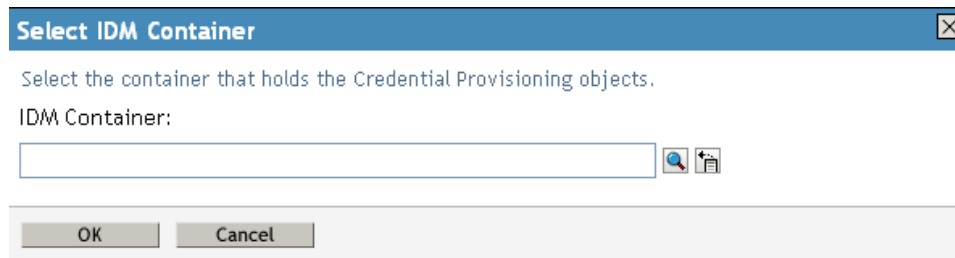
15d Click *OK*.

15e Click the *Save* icon  to save the repository object.

After the repository object is created, proceed to “[Creating an Application Object for Novell SecureLogin](#)” on page 339.

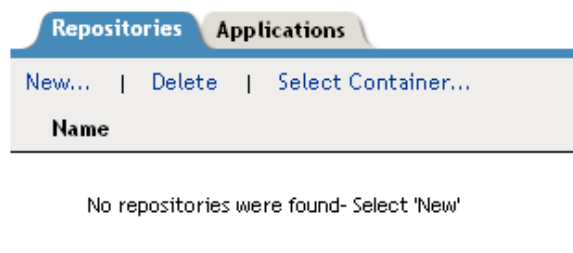
Creating a Repository Object for Novell SecureLogin in iManager

- 1 In iManager, select *Credential Provisioning > Configuration*.
- 2 Browse to and select the Driver object where the repository object will be stored, then click *OK*.

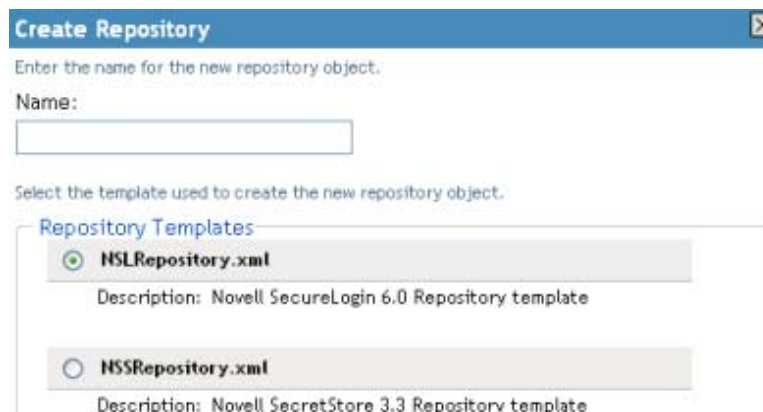


- 3 Click *New* to create a repository.

IDM Container: GroupWise.driverset1.novell



- 4 Specify a name for the repository object, then select *NSLRepository.xml* to use the SecureLogin template to create a repository.



- 5 Click *OK*.

6 Specify the DNS name or IP address of the SecureLogin server. See worksheet item 3).

SecureLogin Server Name or Address ⓘ

7 Specify the SSL port for the SecureLogin server. See worksheet item 4).

SecureLogin Server SSL Port ⓘ

8 Specify the full path to the SSL certificate exported from the SecureLogin server. The path must include the certificate name and must be local to the Identity Manager server. See worksheet item 7).

SecureLogin Server SSL Certificate Path ⓘ

The SecureLogin server can run on multiple platform types. Refer to the platform specific documentation for the steps on how to export the SSL certificate.

9 Specify the fully qualified LDAP distinguished name of the SecureLogin administrator. See worksheet item 5).

SecureLogin Administrator ⓘ

10 Click *Set password*.

SecureLogin Administrator Password ⓘ [Set password](#)

11 Specify the SecureLogin administrator's password twice, then click *OK*. See worksheet item 6).

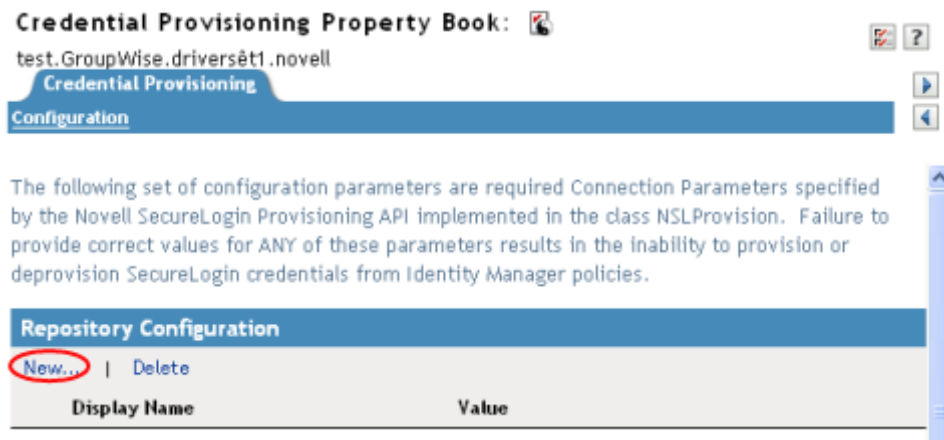
Enter Password ✕

Enter password:

Reenter password:

12 Review the values specified, then click *OK*.

13 (Optional) If you need to create other configuration parameters for the repository, click *New*.



13a Specify a name for the parameter.

13b Specify a display name for the parameter.

13c Specify a description of the parameter for your reference.

The parameter is stored as a string.

Global Configuration Value Definition

Global Configuration Values are a means through which the behavior of an Identity Manager driver configuration can be changed without requiring any policy to be changed.

Name:

Display name:

Description:

Type:

13d Click *OK*.

After the repository object is created, proceed to [“Creating an Application Object for Novell SecureLogin in iManager”](#) on page 342.

4.2.5 Creating an Application Object for Novell SecureLogin

Application objects store application authentication parameter values for SecureLogin. Application information is specific to the applications that are consuming the application credential (for

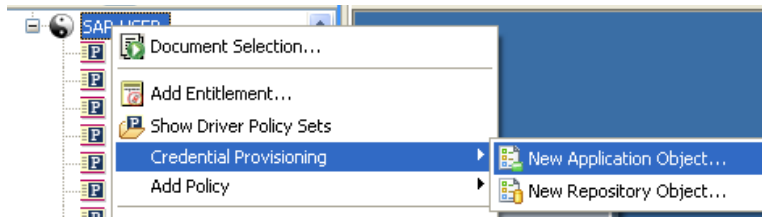
example, GroupWise® client information or SAP database client information). The application objects can be created in Designer or iManager.

- ♦ “Creating an Application Object for Novell SecureLogin in Designer” on page 340
- ♦ “Creating an Application Object for Novell SecureLogin in iManager” on page 342

Creating an Application Object for Novell SecureLogin in Designer

The following is one of many methods you can use to create the application object in Designer.

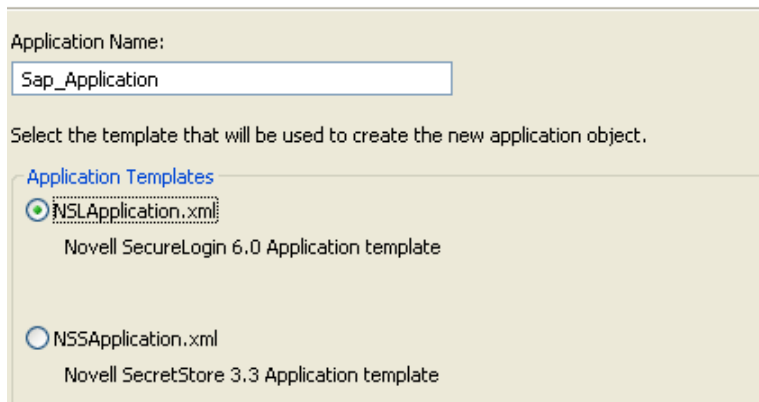
- 1 In the outline view, right-click the driver object where you want to store the application object.
- 2 Click *Credential Provisioning > New Application Object*.



- 3 Specify a name for the application object.
- 4 Select *NSLApplication.xml* to use the SecureLogin template.

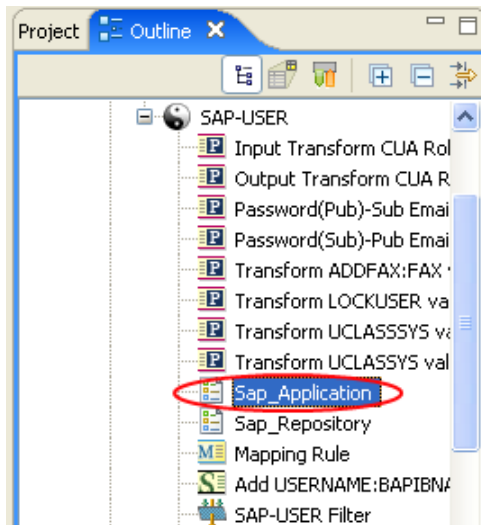
Create Application

Give a name for the application object and select the default template to start with

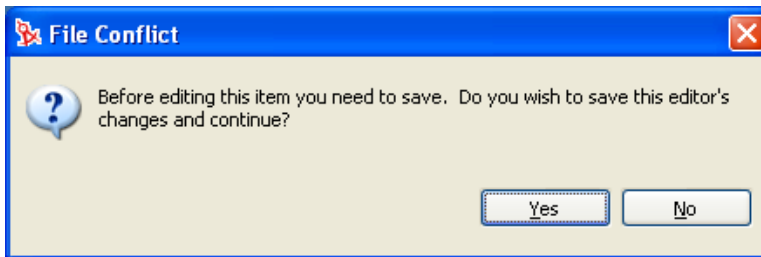
A screenshot of the 'Create Application' dialog box. At the top, there is a label 'Application Name:' followed by a text input field containing the text 'Sap_Application'. Below this, there is a label 'Select the template that will be used to create the new application object.' followed by a section titled 'Application Templates'. Under this section, there are two radio button options. The first option is 'NSLApplication.xml' with a selected radio button, and its description is 'Novell SecureLogin 6.0 Application template'. The second option is 'NSSApplication.xml' with an unselected radio button, and its description is 'Novell SecretStore 3.3 Application template'.

- 5 Click *OK*.

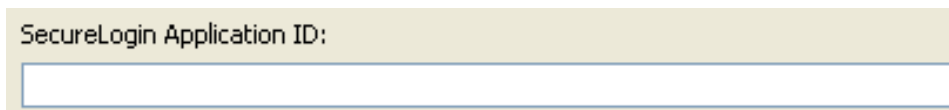
6 Double-click the application object in the outline view to add configuration information.



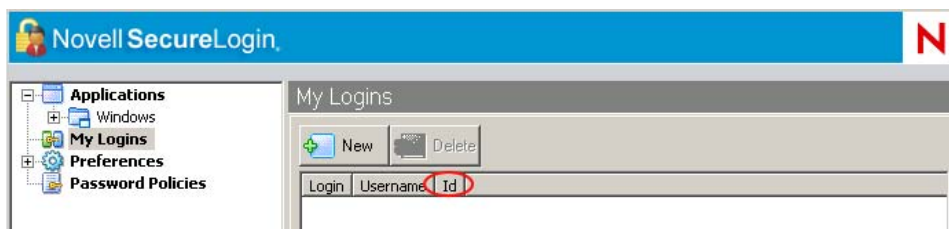
7 Click *Yes* to save the new application object.





8 Specify the SecureLogin Application ID. See worksheet item 9).

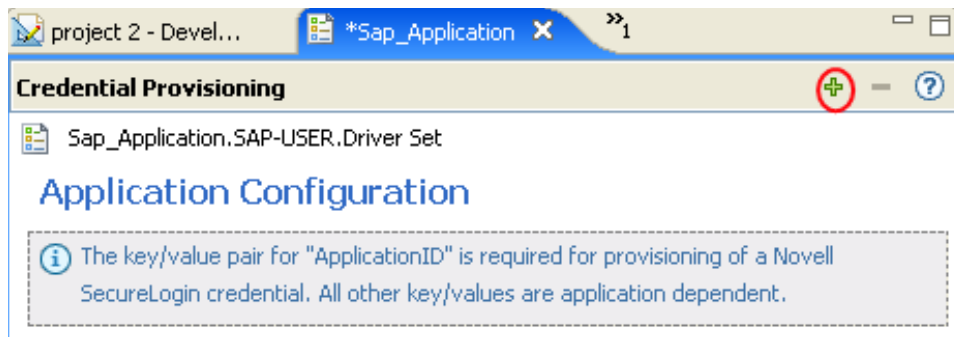


To find the application ID in SecureLogin, click *My Logins*. The application ID is stored in the *Id* field.



9 Click the *Save* icon  to save the application.

- 10 Click the *Add new item* icon  to add the authentication keys required for the application.




- 10a Specify a name for the authentication key.
10b Specify a display name for the authentication key.
10c Specify a description of the authentication key for your reference.
The authentication key is stored as a string.

Name:


Display name:

Description:

Type:
string 

- 10d Click *OK*.
10e Repeat [Step 10](#) for each new authentication key that needs to be entered.

To find the authentication key for your application, manually create a SecureLogin credential for a user in the application and have the user log in. After the user has logged in, the authentication key information is displayed under My Logins in the SecureLogin administration window.

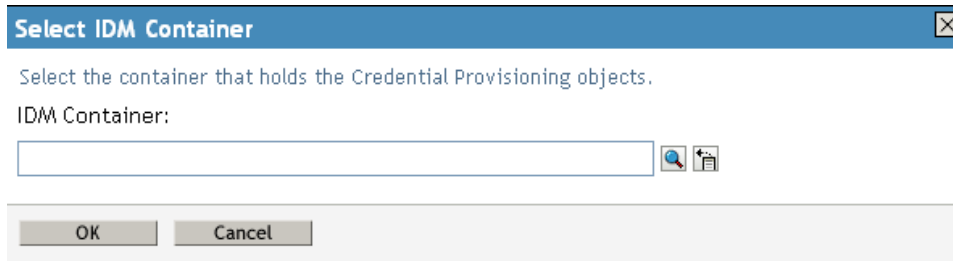
- 11 Specify the authentication key value if it is a static value that is shared by all user credentials.
12 Click the Save icon  to save the application.

After the application object is created, proceed to [“Configuring Credential Provisioning Policies for Novell SecureLogin”](#) on page 345.

Creating an Application Object for Novell SecureLogin in iManager

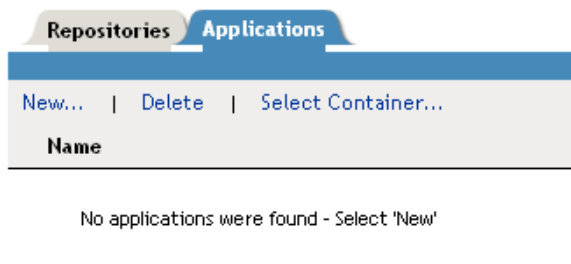
- 1 In iManager, select *Credential Provisioning > Configuration*.

- 2 Browse to and select the Driver object where the application object will be stored, then click *OK*.

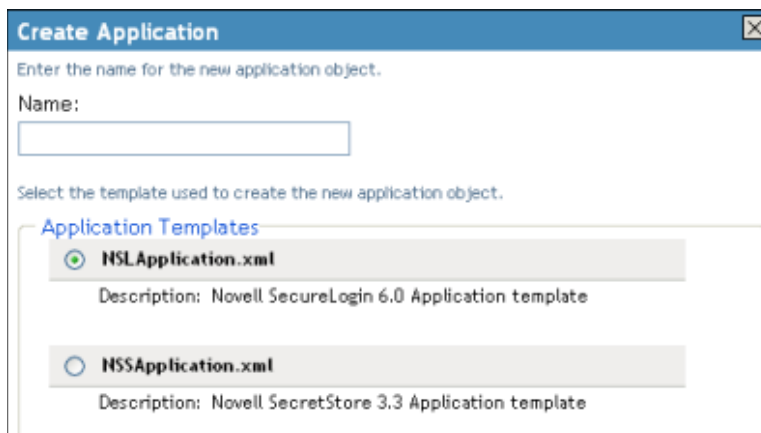


- 3 Select the *Applications* tab, then click *New*.

Container: Delimited Text.DriverSet.Novell



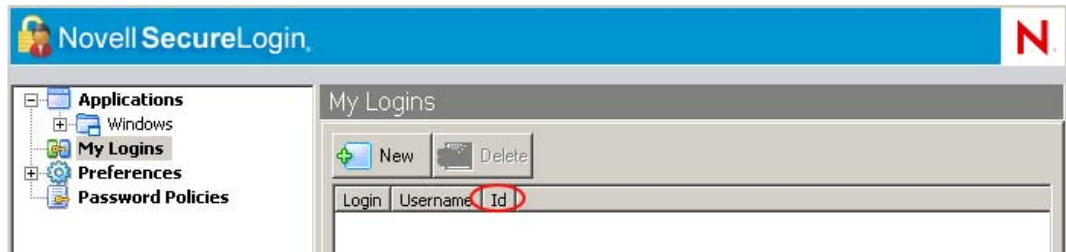
- 4 Specify a name for the application object.
- 5 Select *NSLApplication.xml* to use the SecureLogin template to create an application.



- 6 Click *OK*.
- 7 Specify the *SecureLogin Application ID*. See item worksheet 9).

SecureLogin Application ID ⓘ

To find the application ID in SecureLogin, click *My Logins*. The application ID is stored in the *Id* field.



- 8 Click *New* to create an authentication key parameter. See worksheet item 10).

Credential Provisioning Configuration

The key/value pair for "ApplicationID" is required for provisioning of a Novell SecureLogin credential. All other key/values are application dependent.



- 8a Specify a name for the authentication key.
- 8b Specify a display name for the authentication key.
- 8c Specify a description of the authentication key for your reference.
The authentication key is stored as string.

Global Configuration Value Definition

Global Configuration Values are a means through which the behavior of an Identity Manager driver configuration can be changed without requiring any policy to be changed.

Name:

Display name:

Description:

Type:

To find the authentication key for your application, manually create a SecureLogin credential for a user in the application and have the user log in. After the user has logged

in, the authentication key information is displayed under *My Logins* in the SecureLogin administration window.

8d Click *OK*.

8e Specify the value of the authentication key, if it is static, then click *OK*.

Application Configuration	
New... Delete	
Display Name	Value
<input type="checkbox"/> SecureLogin Application ID ⓘ	SAP - 151.150.191.27
<input type="checkbox"/> Client ⓘ	010
<input type="checkbox"/> Language ⓘ	EN
<input type="checkbox"/> Username ⓘ	
<input type="checkbox"/> Password ⓘ	

After the application object is created, proceed to [“Configuring Credential Provisioning Policies for Novell SecureLogin” on page 345](#).

4.2.6 Configuring Credential Provisioning Policies for Novell SecureLogin

After the repository and application objects are created, policies need to be created to provision SecureLogin information. The policies use the information stored in the repository and application objects. There are three actions in the Policy Builder that allow the provisioning of SecureLogin credentials:

- ◆ [“Clear SSO Credential” on page 346](#)
- ◆ [“Set SSO Credential” on page 346](#)
- ◆ [“Set SSO Passphrase” on page 347](#)

Clear SSO Credential

The *clear SSO credential* action allows you to clear the SSO credential, so objects can be deprovisioned.

Figure 4-2 *Clear SSO Credential*

The screenshot shows the 'Action List' configuration for the 'clear SSO credential' action. The action name is 'clear SSO credential'. Below the name are four input fields: 'Enter credential store object DN:*', 'Enter target user DN:*', 'Enter application credential ID:*', and 'Enter login parameter strings:'. A checkbox labeled 'Render browsed DN relative to policy' is checked. A blue link 'Populate the following from an application object' is located between the 'Enter target user DN:*' and 'Enter application credential ID:*' fields. Each input field has a search icon on the right.

- ◆ **Enter Credential Store Object DN:** Browse to and select the repository object.
- ◆ **Enter Target User DN:** Create the DN of the target users by using the Argument Builder. See worksheet item 15).
- ◆ **Enter Application Credential ID:** Specify the application ID. See worksheet item 9).
- ◆ **Enter Login Parameter Strings:** Launch the String Builder and enter each authentication key for the application. See worksheet item 10).

Set SSO Credential

The *set SSO credential* action allows you to set the SSO credential when a user object is created or when a password is modified.

Figure 4-3 *Set SSO Credential*

The screenshot shows the 'Action List' configuration for the 'set SSO credential' action. The action name is 'set SSO credential'. Below the name are four input fields: 'Enter credential store object DN:*', 'Enter target user DN:*', 'Enter application credential ID:*', and 'Enter login parameter strings:'. A checkbox labeled 'Render browsed DN relative to policy' is checked. A blue link 'Populate the following from an application object' is located between the 'Enter target user DN:*' and 'Enter application credential ID:*' fields. Each input field has a search icon on the right.

- ◆ **Enter Credential Store Object DN:** Browse to and select the repository object.
- ◆ **Enter Target User DN:** Create the DN of the target users by using the Argument Builder. See worksheet item 15).
- ◆ **Enter Application Credential ID:** Specify the application ID. See worksheet item 9).
- ◆ **Enter Login Parameter Strings:** Launch the String Builder and enter each authentication key for the application. See worksheet item 10).

Set SSO Passphrase

The *set SSO passphrase* action allows you to create a SecureLogin passphrase and answer for a user object when it is provisioned.

Figure 4-4 Set SSO Passphrase

The screenshot shows the 'Action List' configuration window. At the top, the action name 'set SSO passphrase' is selected in a dropdown menu. Below this, there are three input fields: 'Enter credential store object DN:*', 'Enter target user DN:*', and 'Enter question and answer strings:*'. A checkbox labeled 'Render browsed DN relative to policy' is checked. Each input field has a search icon to its right.

- ◆ **Enter Credential Store Object DN:** Browse to and select the repository object.
- ◆ **Enter Target User DN:** Create the DN of the target users by using the Argument Builder. See worksheet item 15).
- ◆ **Enter Question and Answer Strings:** Launch the String Builder and enter the passphrase question and answer. See worksheet item 16).

Example Credential Provisioning Policies

The provisioning policies can be implemented and customized to meet the needs of your environment. The following example explains how to implement the policies for the scenario presented in [Figure 4-1 on page 328](#).

In the Finance scenario, SecureLogin provisioning occurs after a password is successfully set in SAP. Most of the necessary parameters are statically configured and available to all policies through the repository and application objects. However, there are non-static data parameters (sapUsername, password, DirXML-ADContext, and workforceID) that are available only after the SAP User Management driver `<add>` or `<modify-password>` commands complete and the `<output>` status document is returned from the SAP User Management driver shim. The `<output>` document no longer contains any of the Subscriber channel operation attributes and the user context of the command is lost, thus preventing queries on the object. It is therefore necessary to do the following:

- ◆ Make sure the SAP User driver's Subscriber Create policy enforces the presence of the non-static data parameters.
- ◆ Cache the non-static parameters required for the provisioning operation prior to issuing the Subscriber command to the SAP User driver shim.
- ◆ Retrieve cached data for use in SecureLogin provisioning after the command completes successfully.

NOTE: Sample policies are available in XML format on the Identity Manager 3.0 Support Pack 1 media. The filenames are `SampleInputTransform.xml`, `SampleSubCommandTransform.xml`, and `SampleSubEventTransform.xml`. The files are found in the following directories, depending upon the platform:

- ◆ `linux\setup\utilities\cred_prov`
- ◆ `nt\dirxml\utilities\cred_prov`
- ◆ `nw\dirxml\utilities\cred_prov`

The files are installed to the Identity Manager server, if Credential Provisioning Sample Policies is selected during the installation of the utilities. The sample policies are installed to the following locations, depending upon the platform:

- ♦ Windows: C:\Novell\NDS\DirXMLUtilities (default; the user can change it during install)
- ♦ NetWare®: SYS:\System\DirXmlUtilities
- ♦ Linux (eDir 8.7): /usr/lib/dirxml/rules/credprov
- ♦ Linux (eDir 8.8.1): /opt/novell/eDirectory/lib/dirxml/rules/credprov (default; the user can change it during the install)

The sample policies provide a starting point to develop a policy that works for your environment.

Operation Data Caching

The mechanism that is available for required operation data caching is the <operation-data> element. Because you might need to provision the SecureLogin account from either an <add> or <modify-password> command, a logical place to implement the non-static data caching policy is in the Subscriber Command Transformation policy. The following example shows a typical SecureLogin Provisioning <operation-data> element:

```
<operation-data>
  <nsl-sync-data>
    <nsl-target-user-dn>
cn=GLCANYON,ou=finance,dc=prod,dc=testco,dc=com
    </nsl-target-user-dn>
    <nsl-app-username>GCANYON</nsl-app-username>
    <password><!-- content suppressed --></password>
    <nsl-passphrase-answer>50024222</nsl-passphrase-answer>
  </nsl-sync-data>
</operation-data>
```

In the sample Finance department scenario from [Figure 4-1 on page 328](#), the following values are needed to populate the operation data payload:

- ♦ The <nsl-target-user-dn> element is populated with the value of the DirXML-ADContext attribute from the Identity Vault, which was set by the Active Directory driver. To ensure that the SAP User driver is notified when the value is set by the AD driver, make sure you add DirXML-ADContext to the Subscriber filter as a notify attribute.
- ♦ The <nsl-app-username> element is populated by the value of the sapUsername attribute which, for an <add> command, is generated by the Create policy of the SAP User driver and is therefore available as an operation attribute. With the SAP User driver, the SAP User name value is part of the association value. This means that for password modification events the names are parsed from the association.
- ♦ The password element is populated with the value of the <password> element in the <add> or <modify-password> command.
- ♦ The <nsl-passphrase-answer> element is populated with the value of the workforceID attribute from the Identity Vault, which was set by the SAP HR driver. Although this value should be set during initial provisioning to the Identity Vault, it is still a good practice to add workforceID to the Subscriber filter as a notify attribute.

SecureLogin Provisioning

In the scenario, the first available location from which the operation data can be retrieved and utilized for SecureLogin credential provisioning is in the driver's Input Transformation policy. In the sample scenario, three policies are implemented:

- ◆ Set SecureLogin Credentials after successful password synchronization.
- ◆ Set SecureLogin Passphrase and Answer
- ◆ Remove SecureLogin Credentials if Application User Deleted (Identity Vault object not deleted)

NOTE: There is a sample policy in the `SampleInputTransform.xml` file that sets SecureLogin credentials after a successful password synchronization occurs. The file is located in the Credential Provisioning folder on the Identity Manager 3.0 Support Pack 1 media.

The Set SecureLogin Credentials policy needs to make sure the provisioning happens only if the returned command status is success and the previously set `<operation-data>` is present.

SecureLogin Deprovisioning

There are many scenarios that can utilize a policy in which a user account for a connected application is deleted and the Identity Vault account remains. In the Finance scenario, there is a requirement to delete the SAP User account and deprovision the SecureLogin credentials when the User's Identity Vault `employeeStatus` attribute value is set to "I". To handle this situation, the SAP User driver's Subscriber Event Transformation contains a policy to transform the modify attribute value into an object delete. Because the Active Directory account name is still needed after the delete command is completed, the `<operation-data>` event needs to be set on the `<delete>` command so it is available to the SecureLogin deprovisioning policy in the Input Transformation policy.

```
<operation-data>
  <nsl-sync-data>
    <nsl-target-user-dn>
cn=GLCANYON,ou=finance,dc=prod,dc=testco,dc=com
    </nsl-target-user-dn>
  </nsl-sync-data>
</operation-data>
```

The policy for transforming the `<modify>` event into a `<delete>` and creating this element is available in the sample Credential Provisioning policies in the `SampleSubEventTransform.xml` file.

4.3 Credential Provisioning Policies with Novell SecretStore

Credential Provisioning policies allow you to provision application credentials to User objects in a Novell SecretStore repository. The capability to provision the Application Server and the User credentials as part of a standard Identity Manager provisioning scenario provides a much more secure and synchronized Web Single Sign-On experience for users.

This document contains the steps required to configure objects and policies in Identity Manager. It does not contain deployment and configuration information for any SecretStore components. For

SecretStore documentation, see [Novell SecretStore 3.3.3 documentation \(http://www.novell.com/documentation/secretstore33/index.html\)](http://www.novell.com/documentation/secretstore33/index.html).

To implement Credential Provisioning with SecretStore requires a repository object, an application object, and creating policies. Repository and application objects store the SecretStore information so that Identity Manager can use it. The policies are used so that any driver can be enabled to use Credential Provisioning. It is also possible to configure the following options:

- Credential Provisioning can be provided by the Publisher channel, Subscriber channel, or both channels.
- SecretStore synchronization can occur as part of an application password synchronization or be triggered by some other event.
- Web Services credentials can be provisioned without provisioning accounts for the application.

Figure 4-5 shows a typical, yet simple, scenario involving the provisioning of the Single Sign-On credentials for a new user in GroupWise®. This department provisions new users into the Identity Vault via a SAP HR system and Identity Manager. Depending on organizational information, the user is then provisioned into a department authentication tree implemented on eDirectory. This is where new users authenticate to the network, and is also the repository of GroupWise security credentials that Novell iChain® or Access Manager® utilizes to provide secure Single Sign-On functionality from outside the company firewall. As users are subsequently provisioned by Identity Manager to GroupWise, the credentials for those systems are synchronized to their SecretStore attributes in the authentication tree.

Figure 4-5 Credential Provisioning with SecretStore

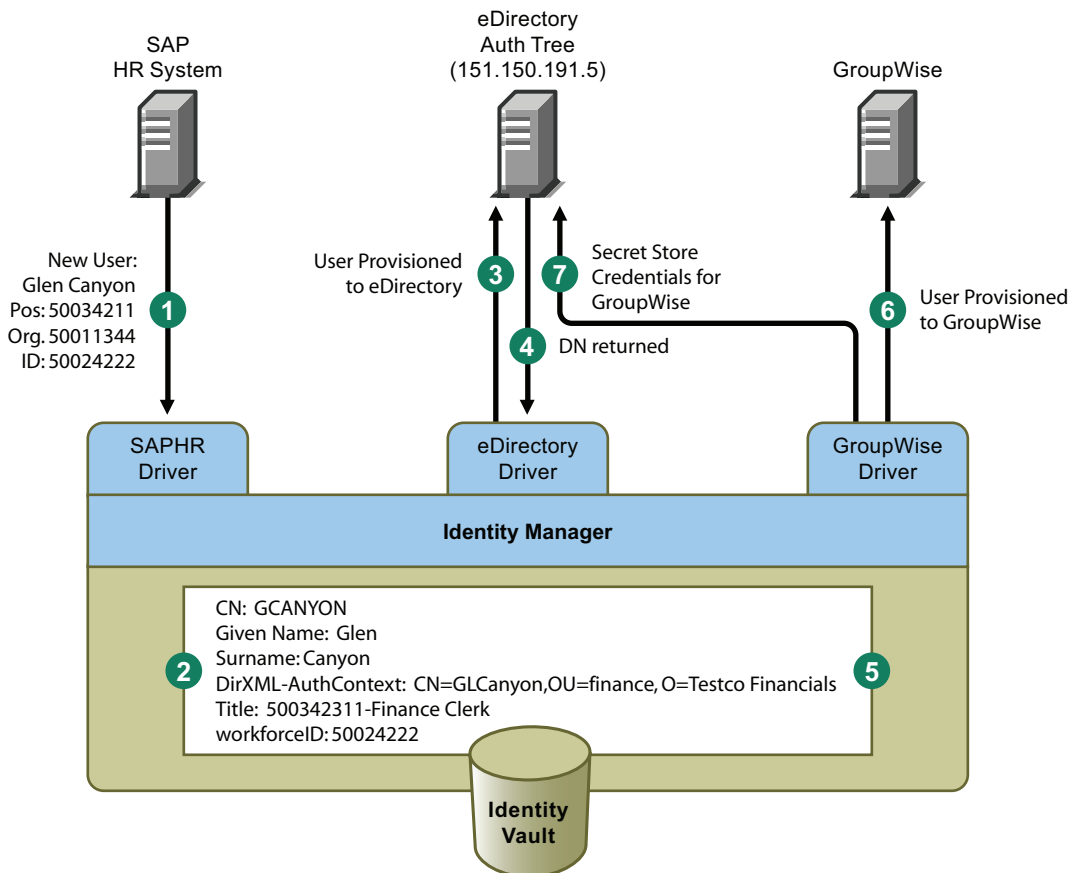


Figure 4-5 illustrates the following provisioning steps:

1. The SAP HR system publishes the data for a newly hired user named Glen Canyon. The Identity Manager SAP HR driver processes this data.
2. A new User object is created in the Identity Vault with a CN value of GCANYON and a workforceID value of 50024222. Because this user is assigned to the Finance organization of his company, he needs to authenticate to the Finance Department eDirectory server. The Identity Manager eDirectory driver that synchronizes that domain now uses the Identity Vault information.
3. Glen is provisioned to the Finance department eDirectory server.
4. The driver is configured to obtain Glen's fully distinguished LDAP name:
CN=GLCanyon,OU=finance,O=Testco Financials.
5. The LDAP name is placed into the DirXML-AuthContext (extension of User object, copy of DirXML-ADContext) attribute of the GCANYON user in the Identity Vault.
Now that the required attributes are available in the Identity Vault, the GroupWise driver processes the attributes of the GCANYON object.
6. Because Glen is in the Finance organization, the driver provisions a GroupWise account for GCANYON on the Finance Departments GroupWise domain server.
7. After the account creation is successful, the GroupWise driver policies provision Glen's GroupWise authentication credentials to the Secret Store of his eDirectory user account.

When Glen authenticates to his company's Web site from the Internet, an iChain server can use the SecretStore credentials to form-fill his authentication to his secure GroupWise e-mail account, eliminating the need for him to enter his GroupWise credentials and also providing additional security for the company's resources.

4.4 Implementing Credential Provisioning Policies with SecretStore

The implementation of Credential Provisioning policies with SecretStore is very customizable. The steps to implement it are different depending upon the platforms SecretStore is installed on, the applications that are provisioned, and which Identity Manager drivers are involved.

To implement Credential Provisioning policies with SecretStore:

- ◆ [Section 4.4.1, "Meeting Requirements for Credential Provisioning Policies with Novell SecretStore," on page 352](#)
- ◆ [Section 4.4.2, "Determining Deployment Configuration Parameters for Novell SecretStore," on page 352](#)
- ◆ [Section 4.4.3, "Creating a Repository Object for Novell SecretStore," on page 355](#)
- ◆ [Section 4.4.4, "Creating an Application Object for Novell SecretStore," on page 361](#)
- ◆ [Section 4.4.5, "Configuring Credential Provisioning Policies for Novell SecretStore," on page 368](#)

4.4.1 Meeting Requirements for Credential Provisioning Policies with Novell SecretStore

In order to use Credential Provisioning Policies with SecretStore, the following items must be in place:

- ◆ Identity Manager 3.0.1
- ◆ Supported on eDirectory 8.7x and eDirectory 8.8.1; eDirectory 8.8 is not supported
- ◆ Verify that `jsso.jar`, `idmcp.jar`, and `jnet.jar` are in the standard location for Identity Manager Java libraries
- ◆ SecretStore 3.3 or above

After you have verified your environment meets the requirements, proceed to [Section 4.4.2, “Determining Deployment Configuration Parameters for Novell SecretStore,”](#) on page 352.

4.4.2 Determining Deployment Configuration Parameters for Novell SecretStore

In order to provide the synchronization functionality described in the deployment scenario illustrated in [Figure 4-5](#), the first step is to gather all of the business process information related to the Identity Manager and SecretStore environments. You can print [Table 4-3, “Credential Provisioning Policies Worksheet for SecretStore,”](#) on page 352, and use it as a worksheet to record the information.

Table 4-3 *Credential Provisioning Policies Worksheet for SecretStore*

Configuration Information Needed	Information
1) Which applications will be configured for Web Single Sign-On provisioning?	
2) The DNS name or IP address of the SecretStore repository server.	
3) The SSL LDAP port for the SecretStore repository server.	
4) The fully qualified LDAP distinguished name of the administrator for the SecretStore repository server.	
5) The password of the administrator for the SecretStore repository server.	
6) The full path and the name of the SSL certificate exported from the SecretStore server. The certificate must be local to the Identity Manager server.	
7) Determine if SecretStore repositories will be used by multiple drivers or if each driver will use a separate repository.	

Configuration Information Needed	Information
8) Record the type of SecretStore secret that is being used.	There are two supported types of secrets: <ul style="list-style-type: none"> ◆ A: Application Secret (SS_App: prefix) ◆ C: Credential Set Secret (SS_CredSet: prefix)
9) The application ID or Credential Set name for each provisioned application.	
10) Find all required authentication keys for each application, such as Username and Password. They might be different for each application.	
11) Determine if any of the authentication key values can be set with a static value.	
12) For non-static values that are or can be different for each user, make a note of the source of the non-static information (event information or Identity Vault attribute values.)	
13) If you are implementing SecretStore provisioning on a driver that is also synchronizing a password to the target application, determine if the SecretStore provisioning takes place before or after the password is set in the target application server.	
14) The name of the Driver object where the repository and application objects are to be stored. (Can be different drivers.)	
15) Determine the DN of the User objects for the target application.	

Example Provisioning Configuration Data

Using the provisioning scenario, the following example data was determined for provision user's SecretStore credentials for the Finance department's GroupWise domain server onto users in the Finance eDirectory authentication tree:

Table 4-4 Example Credential Provisioning Policies Worksheet for SecretStore

Configuration Information Needed	Information
1) Which applications will be configured for Web Single Sign-On provisioning?	GroupWise
2) The DNS name or IP address of the SecretStore repository server.	151.150.191.5
3) The SSL LDAP port for the SecretStore repository server.	636
4) The fully qualified LDAP distinguished name of the administrator for the SecretStore repository server.	cn=admin,ou=finance,o=Tesetco Financials

Configuration Information Needed	Information
5) The password of the administrator for the SecretStore repository server.	dixml
6) The full path and the name of the SSL certificate exported from the SecretStore server. The certificate must be local to the Identity Manager server.	c:\novell\nds\FinanceAD.cer
7) Determine if SecretStore repositories will be used by multiple drivers or if each driver will use a separate repository.	For this example, there is only one repository.
8) Record the type of SecretStore secret that is being used.	There are two supported types of secrets: <ul style="list-style-type: none"> ◆ A: Application Secret (SS_App: prefix) ◆ C: Credential Set Secret (SS_CredSet: prefix)
9) The application ID or Credential Set name for each provisioned application.	GroupWise_Credentials
10) Find all required authentication keys for each application, such as Username and Password. They might be different for each application.	Username Password
11) Determine if any of the authentication key values can be set with a static value.	No static information for this scenario.
12) For non-static values that are or can be different for each user, make a note of the source of the non-static information (event information or Identity Vault attribute values.)	Username: Identity Vault attribute "CN" Password: Event <password>
13) If you are implementing SecretStore provisioning on a driver that is also synchronizing a password to the target application, determine if the SecretStore provisioning takes place before or after the password is set in the target application server.	After
14) The name of the Driver object where the repository and application objects are to be stored. (Can be different drivers.)	GroupWise-Finance driver
15) Determine the DN of the User objects for the target application.	Identity Vault attribute "DirXML-ADContext"

Miscellaneous Environment Information:

- ◆ The Finance department eDirectory tree would serve as the SecretStore repository for all Finance applications.
- ◆ All finance department provisioning drivers are in a driver set call Finance Drivers.
- ◆ The GroupWise account must be deleted and the SecretStore credentials for the GroupWise user account must be removed from the eDirectory user when the Identity Vault attribute employeeStatus is set to the value "I".

As can be seen from the data gathered, the SecretStore repository information is global for all drivers that provision Finance department applications. In addition all provisioning information can

be statically configured, with the exception of the GroupWise login parameters Username, Password, and Target User DN.

After all of the configuration data has been determined, proceed to [Section 4.4.3, “Creating a Repository Object for Novell SecretStore,” on page 355.](#)

4.4.3 Creating a Repository Object for Novell SecretStore

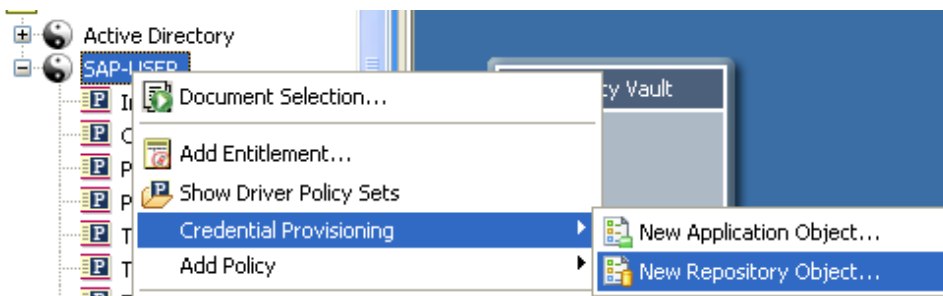
Repository objects store static configuration information for SecretStore. Repository information is independent from the applications that consume the application credentials. This information is applicable for all provisioning events regardless of the connected system (for example SAP, PeopleSoft, Notes, etc.) The repository objects can be created in Designer or iManager.

- ♦ [“Creating Repository Objects for Novell SecretStore in Designer” on page 355](#)
- ♦ [“Creating Repository Objects for Novell SecretStore in iManager” on page 358](#)

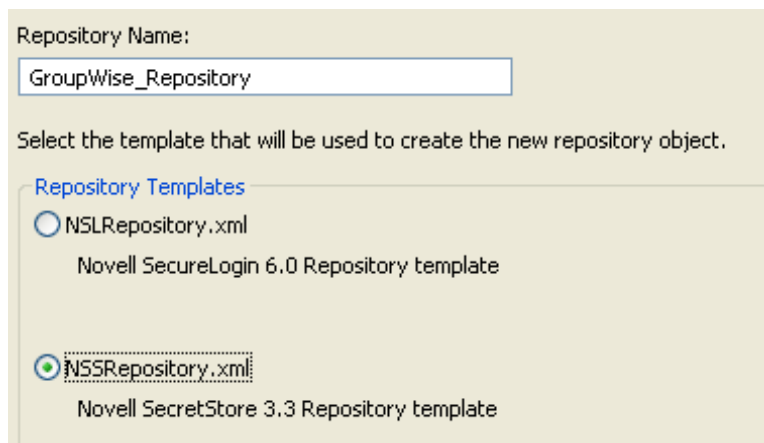
Creating Repository Objects for Novell SecretStore in Designer

The following is one of many methods you can use to create the repository object in Designer.

- 1 In the outline view, right-click the driver object where you want to store the repository object.
- 2 Click *Credential Provisioning > New Repository Object*.

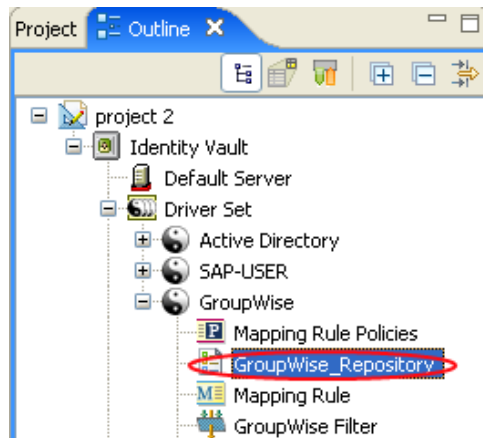


- 3 Specify a name for the repository object.
- 4 Select *NSSRepository.xml* to use the SecretStore template.

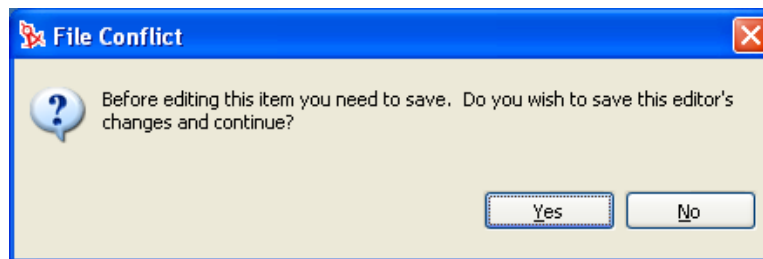


- 5 Click *OK*.

- 6 Double-click the repository object in the outline view to add configuration information.




- 7 Click *Yes*, to save the new repository object.



- 8 Specify the DNS name or IP address of the SecretStore server. See worksheet item 2).

SecretStore Server Name or Address:

- 9 Specify the SSL port for the SecretStore server. See worksheet item 3).

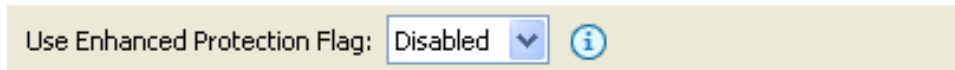
SecretStore Server SSL Port: 

- 10 Specify the full path to the SSL certificate exported from the SecretStore server. The path must include the certificate name and must be local to the Identity Manager server. See worksheet item 6).

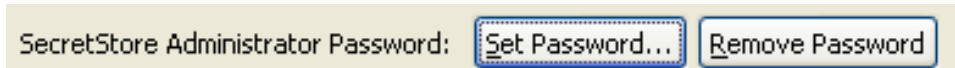
SecretStore Server SSL Certificate Path:

NOTE: Refer to the iManager documentation for the information on how to export the SSL certificate.

- 11 Specify the fully qualified LDAP distinguished name of the SecretStore administrator. See worksheet item 4).





- 12 Click *Set password*.

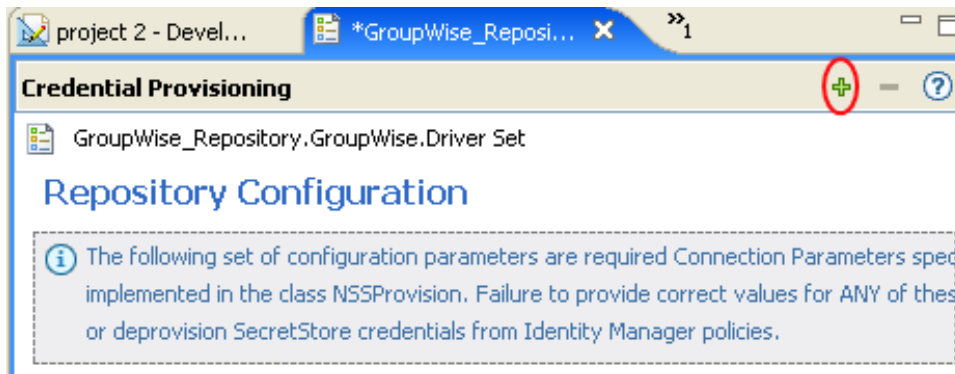


- 13 Specify the SecretStore administrator's password twice, then click OK. See worksheet item 5).



- 14 Review the information, then click the *Save* icon  to save the information.

- 15 (Optional) If you want to create other configuration parameters for the repository object, click the *Add new item* icon .



- 15a Specify a name for the parameter.

- 15b Specify a display name for the parameter.

- 15c Specify a description for the parameter for your reference.

The parameter is stored as a string.

Name:

Display name:

Description:

Type:
string ▼

15d Click *OK*.

15e Click the *Save* icon  to save the repository object.



After the repository object is created, proceed to [“Creating an Application Object for Novell SecureLogin in Designer” on page 340](#).

Creating Repository Objects for Novell SecretStore in iManager

- 1 In iManager, select *Credential Provisioning > Configuration*.
- 2 Browse to and select the Driver object where the repository object will be stored, then click *OK*.

Select IDM Container ✕

Select the container that holds the Credential Provisioning objects.

IDM Container:
  

OK Cancel

- 3 Click *New* to create a repository.

IDM Container: GroupWise.driversnovell

Repositories Applications

New... | Delete | Select Container...

Name

No repositories were found- Select 'New'

- 4 Specify a name for the repository object.
- 5 Select *NSSRepository.xml* to use the SecretStore template to create a repository.

- 6 Click *OK*.
- 7 Specify the DNS name or IP address of the SecretStore server. See worksheet item 2).

SecretStore Server Name or Address ⓘ

- 8 Specify the SSL port for the SecretStore server. See worksheet item 3).

SecretStore Server SSL Port ⓘ

- 9 Specify the full path to the SSL certificate exported from the SecretStore server. The path must include the certificate name and must be local to the Identity Manager server. See worksheet item 6).

SecretStore Server SSL Certificate Path ⓘ

NOTE: Refer to the iManager documentation for the information on how to export the SSL certificate.

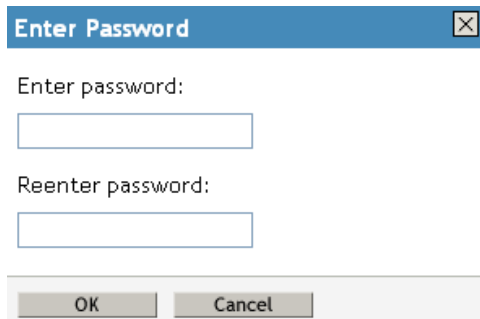
- 10 Specify the fully qualified LDAP distinguished name of the SecretStore administrator. See worksheet item 4).

SecretStore Administrator ⓘ

- 11 Click *Set password*.

SecretStore Administrator Password ⓘ [Set password](#)

- 12 Specify the SecretStore administrator's password twice, then click *OK*. See worksheet item 5).



Enter Password

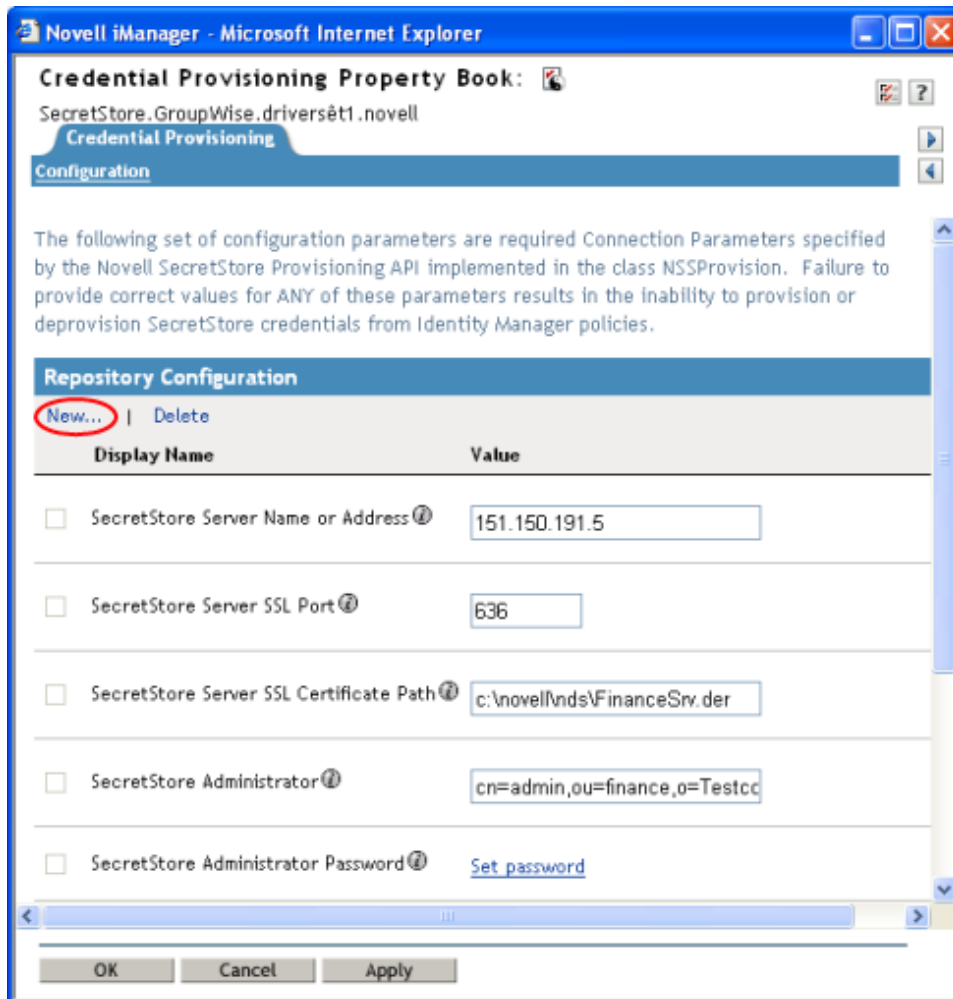
Enter password:

Reenter password:

OK Cancel

- 13 Review the values specified, then click *OK*.

- 14 (Optional) If you want to create other configuration parameters for the repository object, click *New*.



Novell iManager - Microsoft Internet Explorer

Credential Provisioning Property Book: SecretStore.GroupWise.driverset1.novell

Credential Provisioning

Configuration

The following set of configuration parameters are required Connection Parameters specified by the Novell SecretStore Provisioning API implemented in the class NSSProvision. Failure to provide correct values for ANY of these parameters results in the inability to provision or deprovision SecretStore credentials from Identity Manager policies.

Repository Configuration

New... | Delete

Display Name	Value
<input type="checkbox"/> SecretStore Server Name or Address	151.150.191.5
<input type="checkbox"/> SecretStore Server SSL Port	636
<input type="checkbox"/> SecretStore Server SSL Certificate Path	c:\novell\nds\FinanceSrv.der
<input type="checkbox"/> SecretStore Administrator	cn=admin,ou=finance,o=Testcc
<input type="checkbox"/> SecretStore Administrator Password	Set password

OK Cancel Apply

The example information, is from the scenario in [Figure 4-1 on page 328](#).

- 14a Specify a name for the parameter.

- 14b Specify a display name for the parameter.

14c Specify a description of the parameter for your reference.

The parameter is stored as a string.

Global Configuration Value Definition

Global Configuration Values are a means through which the behavior of an Identity Manager driver configuration can be changed without requiring any policy to be changed.

Name:

Display name:

Description:

Type:
string

14d Click *OK*.

After the repository object is created, proceed to [“Creating an Application Object for Novell SecureLogin in iManager” on page 342](#).

4.4.4 Creating an Application Object for Novell SecretStore

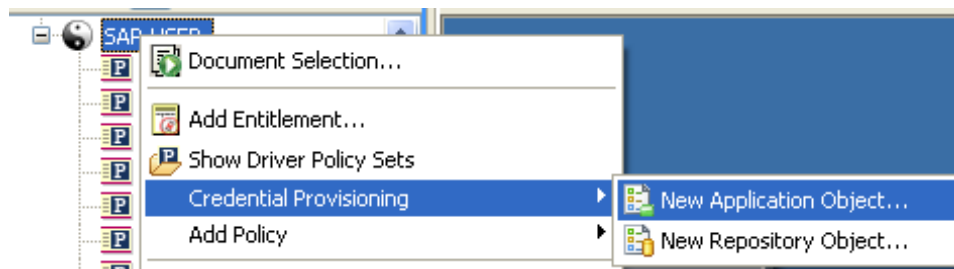
Applications store static configuration parameter values for SecretStore. Application information is specific to the applications that are consuming the application credential (for example, GroupWise client information or SAP database client information). The application objects can be created in Designer or iManager.

- ♦ [“Creating an Application Object for Novell SecretStore in Designer” on page 361](#)
- ♦ [“Creating an Application Object for Novell SecretStore in iManager” on page 364](#)

Creating an Application Object for Novell SecretStore in Designer

The following is one of many methods you can use to create the application in Designer.

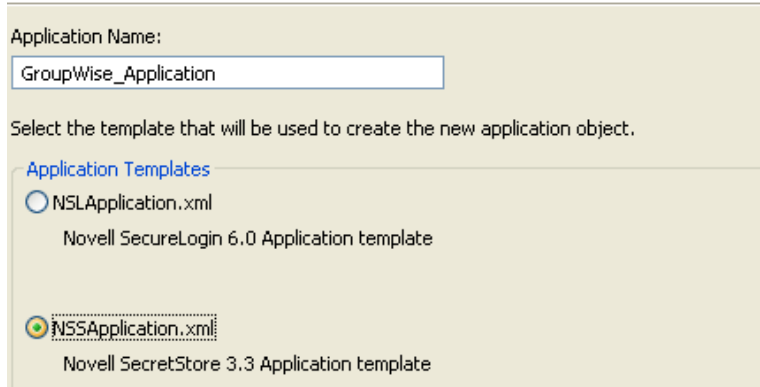
- 1** In the outline view, right-click the driver object where you want to store the application object.
- 2** Click *Credential Provisioning > New Application Object*.



- 3 Specify a name for the application object.
- 4 Select *NSSApplication.xml* to use the SecretStore template.

Create Application

Give a name for the application object and select the default template to start with



Application Name:
GroupWise_Application

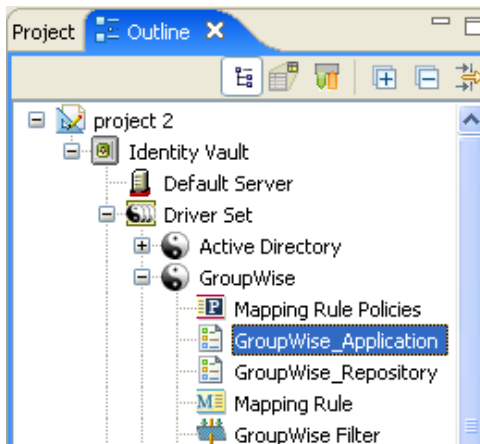
Select the template that will be used to create the new application object.

Application Templates

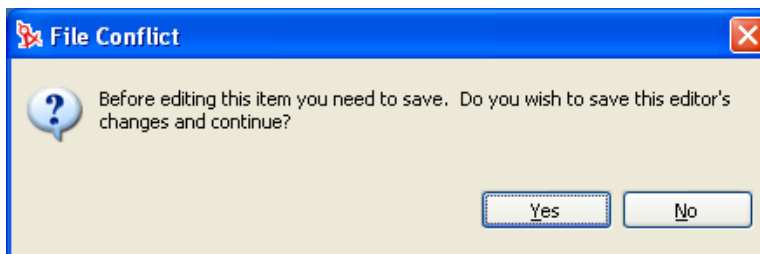
NSLApplication.xml
Novell SecureLogin 6.0 Application template

NSSApplication.xml
Novell SecretStore 3.3 Application template

- 5 Click *OK*.
- 6 Double-click the application object in the outline view to add configuration information.



- 7 Click *Yes* to save the new application object.




- 8 Specify the SecretStore Application ID. See worksheet item 9).

SecretStore Application ID:
GroupWise_Credentials


- 9 Select the *SecretStore Secret Type*. See worksheet item 8).

SecretStore Secret Type: Shared 


- 10 Select the *SecretStore Shared Secret Type*. See worksheet item 8).

SecretStore Shared Secret Type: Credential Set 

- 11 Select whether the *SecretStore Use Enhanced Protection Flag* is Disabled or Enabled.

Use Enhanced Protection Flag: Disabled 

- 12 Click *Set Password* to set the *Enhanced Protection Password* if it is enabled.


Enhanced Protection Password: 

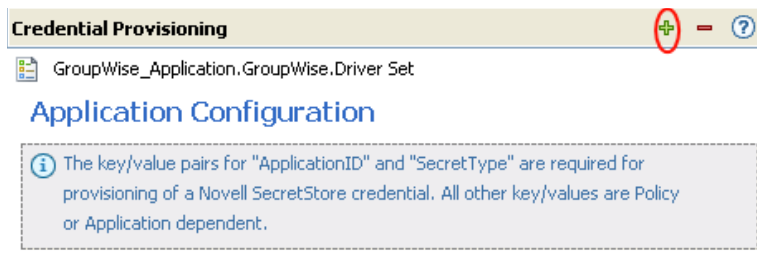
- 13 Specify the password twice, then click *OK*.



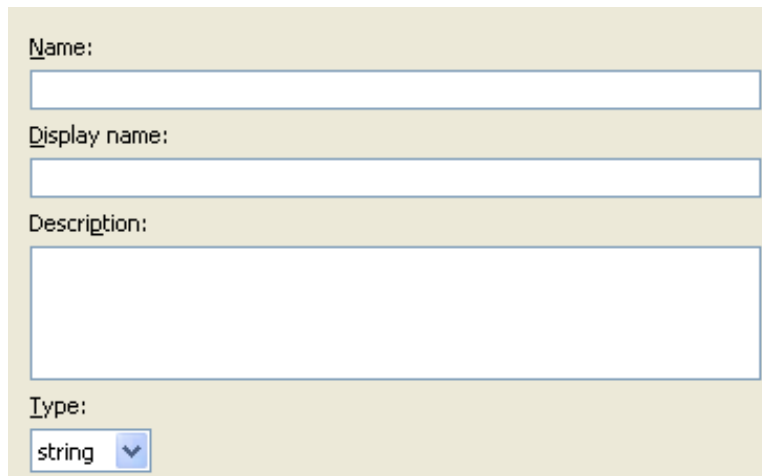
The image shows a 'Change Password' dialog box with a blue title bar and a close button. It contains two text input fields: 'Enter password:' and 'Re-enter password:'. Both fields contain seven black dots representing a password. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

- 14 Click the *Save* icon  to save the application.

- 15** Click the *Add new item* icon  to add the authentication keys required for the application.




- 15a** Specify a name for the authentication key.
15b Specify a display name for the authentication key.
15c Specify a description of the authentication key for your reference.
The authentication key is stored as a string.




Name:

Display name:

Description:

Type:
string 

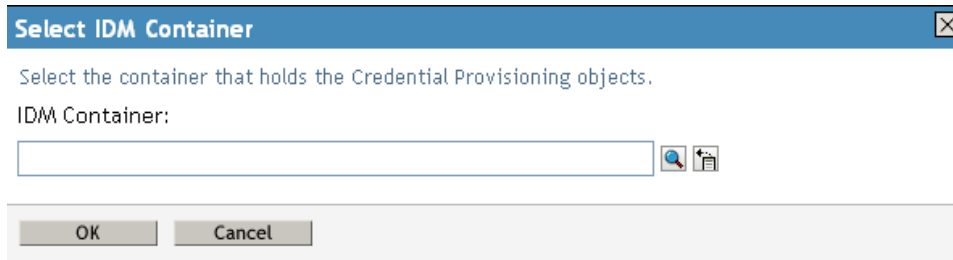
- 15d** Click *OK*.
15e Repeat [Step 15](#) for each new authentication key that needs to be entered.
16 Specify the authentication key value, if it is a static value that is shared by all user credentials.
17 Click the *Save* icon  to save the application.

After the application object is created, proceed to [Section 4.4.5, “Configuring Credential Provisioning Policies for Novell SecretStore,”](#) on page 368.

Creating an Application Object for Novell SecretStore in iManager

- 1** In iManager, select *Credential Provisioning > Configuration*.

- 2 Browse to and select the Driver object where the application object will be stored, then click *OK*.



Select IDM Container [X]

Select the container that holds the Credential Provisioning objects.

IDM Container:

- 3 Select the *Applications* tab, then click *New*.

Container: Delimited Text.DriverSet.Novell

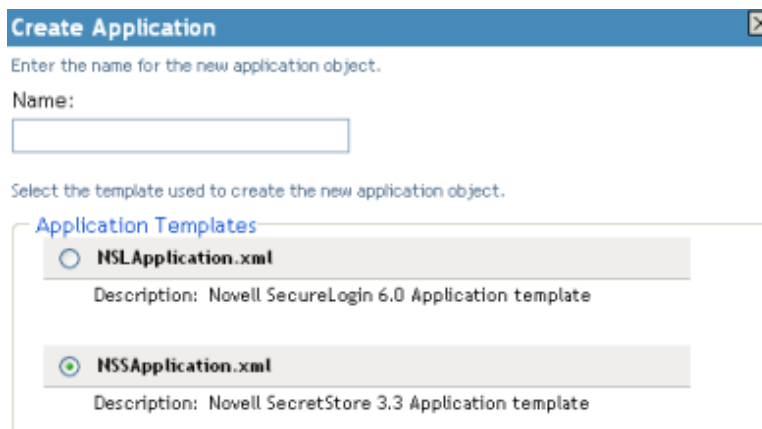


Repositories **Applications**

New... | Delete | Select Container...

Name
No applications were found - Select 'New'

- 4 Specify a name for the application object
- 5 Select *NSSApplication.xml* to use the SecretStore template to create an application.



Create Application [X]

Enter the name for the new application object.

Name:

Select the template used to create the new application object.

Application Templates

- NSLApplication.xml**
Description: Novell SecureLogin 6.0 Application template
- NSSApplication.xml**
Description: Novell SecretStore 3.3 Application template

- 6 Click *OK*.
- 7 Specify the *SecretStore Application ID*. See worksheet item 9).

SecretStore Application ID ⓘ

8 Select the *SecretStore Secret Type*. See worksheet item **7**). The SecretStore type is *Shared* or *Not Shared*.

SecretStore Secret Type ⓘ Shared ▼

9 Select the *SecretStore Shared Secret Type*. See worksheet item **8**). The Shared SecretStore type is *Credential Set* or *Application*.

SecretStore Shared Secret Type ⓘ Credential Set ▼

10 Select whether the SecretStore *Use Enhanced Protection Flag* is *Disabled* or *Enabled*.

Use Enhanced Protection Flag ⓘ Disabled ▼

11 Click *Set password* to set the *Enhanced Protection Password* if it is enabled.

Enhanced Protection Password ⓘ [Set password](#)

12 Specify the password twice, then click *OK*.

Enter Password [X]

Enter password:

Reenter password:

OK Cancel

13 Click *New* to create an authentication key that the application requires. See worksheet item **10**).

13a Specify a name for the authentication key.

13b Specify a display name for the authentication key.

13c Specify a description of the authentication key for your reference.

The authentication key is stored as a string.

Global Configuration Value Definition

Global Configuration Values are a means through which the behavior of an Identity Manager driver configuration can be changed without requiring any policy to be changed.

Name:

Display name:

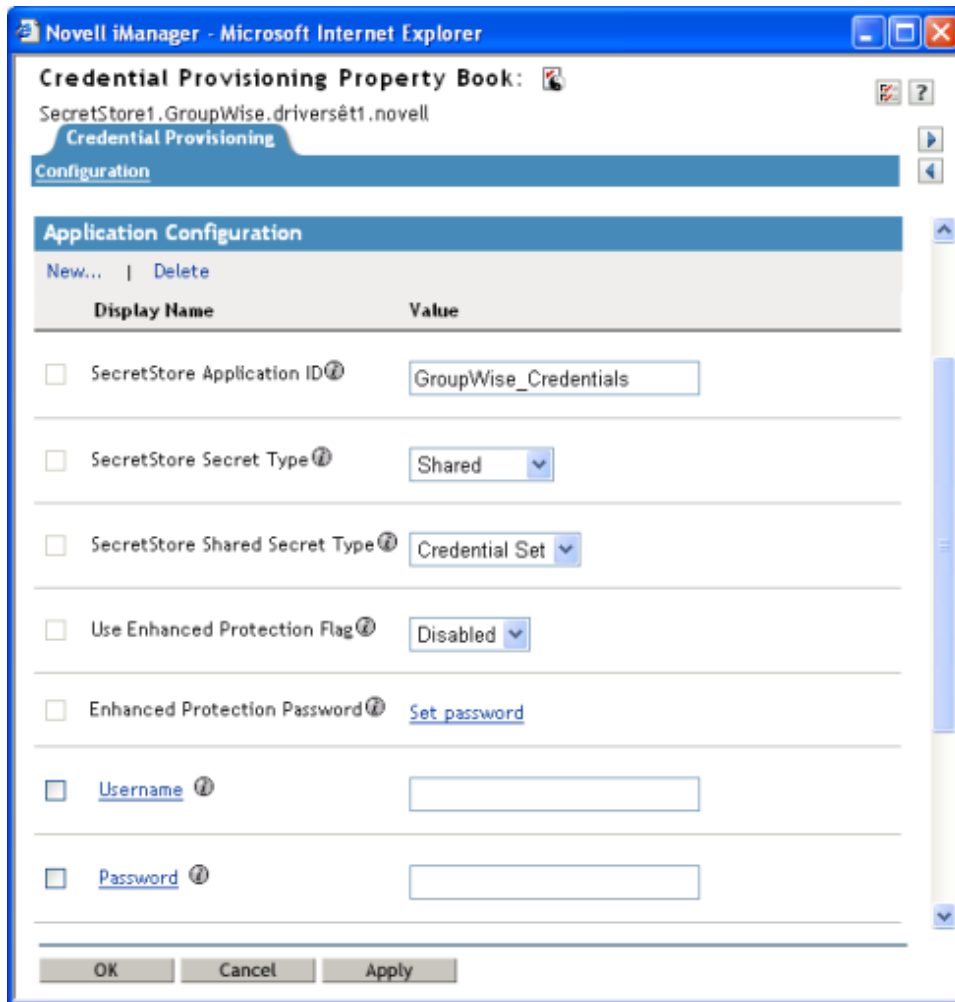
Description:

Type:

13d Click *OK*.

13e Repeat [Step 13](#) for each authentication key the application requires.

14 Specify the value of the authentication key, if it is static, then click *OK*.



After the application object is created, proceed to [Section 4.4.5, “Configuring Credential Provisioning Policies for Novell SecretStore,”](#) on page 368.

4.4.5 Configuring Credential Provisioning Policies for Novell SecretStore

After the repository and application objects are created, policies need to be created to provision SecretStore information. The policies use the information stored in the repository and application objects. There are two actions in the Policy Builder that allow the provisioning of SecretStore credentials:

- ◆ [“Clear SSO Credential”](#) on page 346
- ◆ [“Set SSO Credential”](#) on page 346

Clear SSO Credential

The *clear SSO credential* action allows you to clear the SSO credential, so objects can be deprovisioned.

Figure 4-6 Clear SSO Credential

The screenshot shows the 'Action List' configuration for the 'clear SSO credential' action. The action name is selected in a dropdown menu. Below it, there are four input fields: 'Enter credential store object DN:*', 'Enter target user DN:*', 'Enter application credential ID:*', and 'Enter login parameter strings:'. A checkbox labeled 'Render browsed DN relative to policy' is checked. A link 'Populate the following from an application object' is visible below the 'Enter target user DN:*' field.

- ◆ **Enter Credential Store Object DN:** Browse to and select the repository object.
- ◆ **Enter Target User DN:** Create the DN of the target users by using the Argument Builder. See worksheet item 15).
- ◆ **Enter Application Credential ID:** Specify the application ID. See worksheet item 9).
- ◆ **Enter Login Parameter Strings:** Launch the String Builder and enter each authentication key for the application. See worksheet item 10).

Set SSO Credential

The *set SSO credential* action allows you to set the SSO credential when a user object is created or when a password is modified.

Figure 4-7 Set SSO Credential

The screenshot shows the 'Action List' configuration for the 'set SSO credential' action. The action name is selected in a dropdown menu. Below it, there are four input fields: 'Enter credential store object DN:*', 'Enter target user DN:*', 'Enter application credential ID:*', and 'Enter login parameter strings:'. A checkbox labeled 'Render browsed DN relative to policy' is checked. A link 'Populate the following from an application object' is visible below the 'Enter target user DN:*' field.

- ◆ **Enter Credential Store Object DN:** Browse to and select the repository object.
- ◆ **Enter Target User DN:** Create the DN of the target users by using the Argument Builder. See worksheet item 15).
- ◆ **Enter Application Credential ID:** Specify the application ID. See worksheet item 9).
- ◆ **Enter Login Parameter Strings:** Launch the String Builder and enter each authentication key for the application. See worksheet item 10).

Example Credential Provisioning Policies

The credential provisioning policies can be implemented and customized to meet the needs of your environment. The following example explains how to implement the policies for the scenario presented in [Figure 4-5 on page 350](#).

In the Finance scenario, SecretStore provisioning occurs after a password is successfully set in GroupWise. Most of the necessary parameters are statically configured and available to all policies through the repository and application objects. However, there are non-static data parameters (CN, password, and DirXML-ADContext) that are available only after the GroupWise user `<add>` or `<modify-password>` commands complete and the `<output>` document is returned from the GroupWise driver shim. The `<output>` document no longer contains any of the Subscriber operation attributes and the User context of the command is lost, thus preventing queries on the object. It is therefore necessary to do the following:

- ◆ Make sure the GroupWise driver's Subscriber Create policy enforces the presence of the non-static data parameters.
- ◆ Cache the non-static parameters required for the provisioning operation prior to issuing the Subscriber command to the GroupWise driver shim.
- ◆ Retrieve cached data for use in SecretStore provisioning after the command completes successfully.

NOTE: Sample policies are available in XML format on the Identity Manager 3.0 Support Pack 1 media. The filenames are `SampleInputTransform.xml`, `SampleSubCommandTransform.xml`, and `SampleSubEventTransform.xml`. The files are found in the following directories:

- ◆ `linux\setup\utilities\cred_prov`
- ◆ `nt\dirxml\utilities\cred_prov`
- ◆ `nw\dirxml\utilities\cred_prov`

The files are installed to the Identity Manager server, if Credential Provisioning Sample Policies is selected during the installation of the utilities. The sample policies are installed to the following locations, depending upon the platform:

- ◆ Windows: `C:\Novell\NDS\DirXMLUtilities` (default; the user can change it during install)
- ◆ NetWare: `SYS:\System\DirXmlUtilities`
- ◆ Linux (eDir 8.7): `/usr/lib/dirxml/rules/credprov`
- ◆ Linux (eDir 8.8.1): `/opt/novell/eDirectory/lib/dirxml/rules/credprov` (default; the user can change it during the install)

The sample policies provide a starting point to develop a policy that works for your environment.

Operation Data Caching

The mechanism that is available for required operation data caching required is the `<operation-data>` element. Because you might need to provision the SecretStore account from either an `<add>` or `<modify-password>` command, a logical place to implement the non-static data caching policy is in the Subscriber Command Transformation policy. The following example shows a typical SecretStore Provisioning element:


```

<operation-data>
  <nss-sync-data>
    <nss-target-user-dn>
cn=GLCANYON,ou=finance,o=Testco Financials
    </nss-target-user-dn>
    <nss-app-username>GCANYON</nsl-app-username>
    <password><!-- content suppressed --></password>
    <nss-passphrase-answer>50024222</nsl-passphrase-answer>
  </nss-sync-data>
</operation-data>

```

In the sample Finance department scenario from [Figure 4-5 on page 350](#), the following values are needed to populate the operation data payload:

- ◆ The `<nss-target-user-dn>` element is populated with the value of the DirXML-ADContext attribute from the Identity Vault, which was set by the eDirectory driver. To ensure that the GroupWise driver is notified when the value is set by the eDirectory driver, make sure you add DirXML-ADContext to the Subscriber filter as a notify attribute.
- ◆ The `<nss-app-username>` element is populated by the value of the CN attribute in the Identity Vault.
- ◆ The password element is populated with the value of the `<password>` element in the `<add>` or `<modify-password>` command.

SecretStore Provisioning

In the sample scenario, the first available location from which the operation data can be retrieved and utilized for SecretStore credential provisioning is in the driver's Input Transformation policy. In the sample scenario, two policies are implemented:

- ◆ Set SecretStore Credentials after successful password synchronization
- ◆ Remove SecretStore Credentials if Application User Deleted (Identity Vault object not deleted)

NOTE: There is a sample policy in the `SampleInputTransform.xml` file that sets the SecretStore credentials after a successful password synchronization occurs. The file is located in the `cred_prov` folder in the utilities directory on the Identity Manager 3.0 Support Pack 1 media.

The Set SecretStore Credentials policy needs to make sure the provisioning happens only if the returned command status is Success and the previously set `<operation-data>` is present.

SecretStore Deprovisioning

There are many scenarios that can utilize a policy in which a user account for a connected application is deleted and the Identity Vault account remains. In the Finance scenario, there is a requirement to delete the GroupWise account and deprovision the SecretStore credentials when the user's Identity Vault `employeeStatus` attribute value is set to "I". To handle this situation, the GroupWise driver's Subscriber Event Transformation contains a policy to transform the modify attribute value into an object delete. Because the eDirectory account name is still needed after the delete command is completed, the `<operation-data>` event needs to be set on the `<delete>` command so it is available to the SecretStore deprovisioning policy in the Input Transformation policy.

```
<operation-data>
  <nss-sync-data>
    <nss-target-user-dn>cn=GLCANYON,ou=finance,o=Testco
Financials
    </nss-targer-user-dn>
  </nss-sync-data>
</operation-data>
```

The policy for transforming the <modify> event into a <delete> and creating this element is available in XML format in a file called `SampleSubEventTransform.xml` files in the `cred_prov` folder in the `utilities` directory on the Identity Manager 3.0 Support Pack 1 media.

Defining Policies using XSLT Style Sheets

5

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 5.1, “Managing XSLT Style Sheets in Designer,” on page 373](#)
- ♦ [Section 5.2, “Managing XSLT Style Sheets in iManager,” on page 375](#)
- ♦ [Section 5.3, “Starting with an Identity Transformation,” on page 376](#)
- ♦ [Section 5.4, “Using the Parameters that Identity Manager Passes,” on page 377](#)
- ♦ [Section 5.5, “Using Extension Functions,” on page 379](#)
- ♦ [Section 5.6, “Creating a Password Example: Creation Policy,” on page 380](#)
- ♦ [Section 5.7, “Creating an eDirectory User Example: Creation Policy,” on page 381](#)

5.1 Managing XSLT Style Sheets in Designer

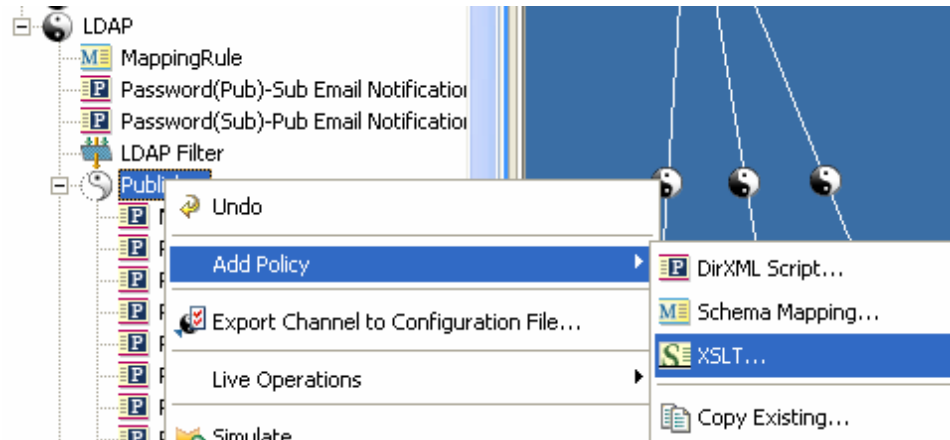
XSLT policy style sheets are added, modified, and deleted using Designer. The following sections provide details on using XSLT style sheets in Designer:

- ♦ [Section 5.1.1, “Adding an XSLT Policy in Designer,” on page 373](#)

5.1.1 Adding an XSLT Policy in Designer

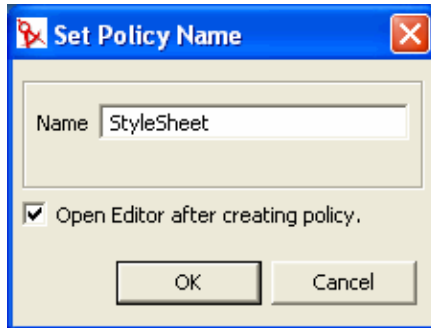
- 1 Open a project in Designer and select the Outline tab.
- 2 Select the driver and location where you want the style sheet.

3 Right-click and select *Add Policy >XSLT*.

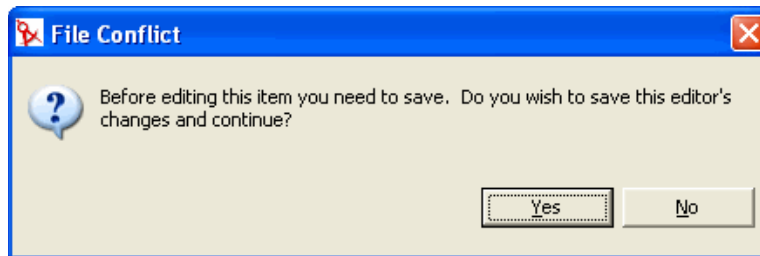


4 Specify the name of the style sheet.


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
  <!-- the stylesheet to 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*.

5.2 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:

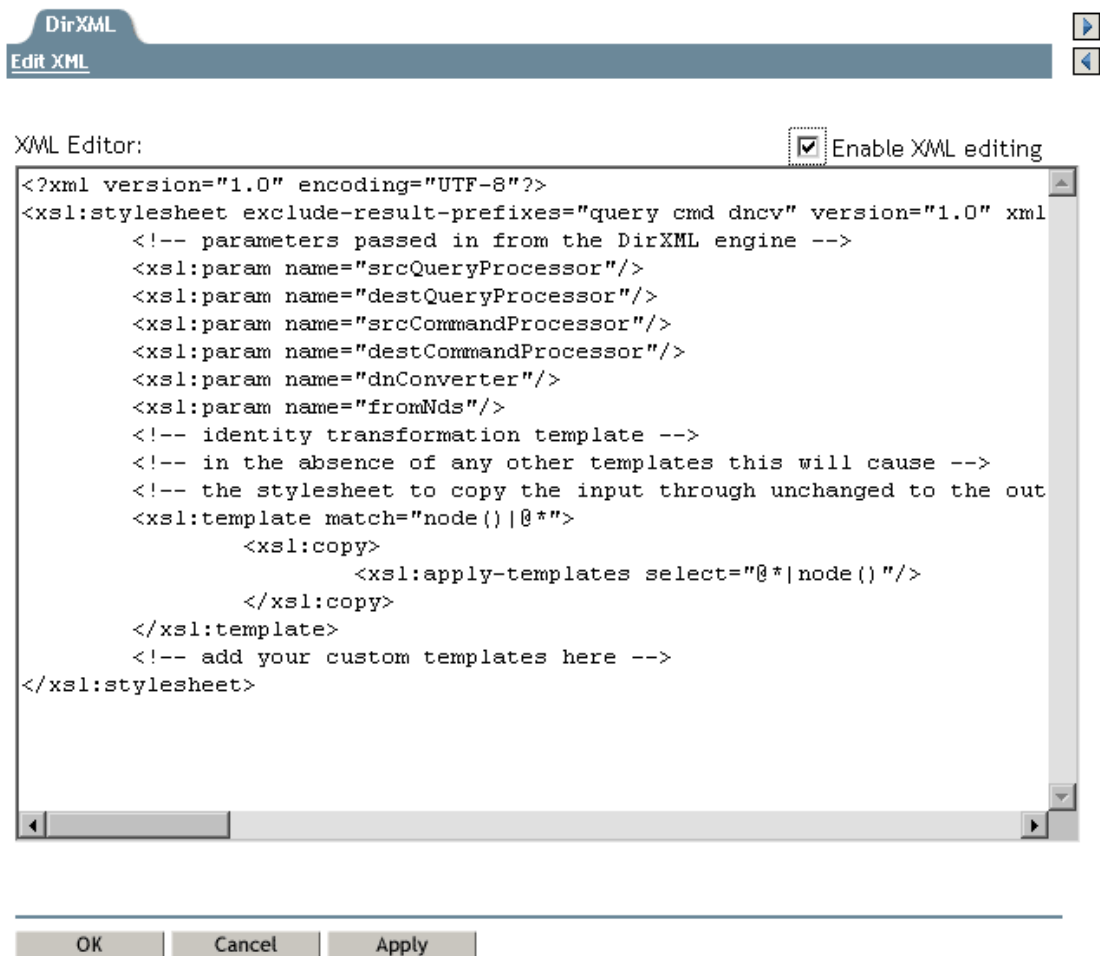
- ♦ [Section 5.2.1, “Adding an XSLT Policy in iManager,” on page 375](#)

5.2.1 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*:

DirXML Policy:  xslt policy



5.3 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.

5.4 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}"/>

      <!-- 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>

```

5.5 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>

```

5.6 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>
```

5.7 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: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>
```


The Filter editor allows you to manage the filter. In the Filter editor, you define how each class and attribute should be handled by the Publisher and Subscriber channels.

This section covers the following filter-related topics:

- ♦ [Section 6.1, “Filter Tasks in Designer,” on page 387](#)
- ♦ [Section 6.2, “Filter Tasks in iManager,” on page 408](#)

6.1 Filter Tasks in Designer


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

- ♦ [Section 6.1.1, “Accessing the Filter Editor,” on page 387](#)
- ♦ [Section 6.1.2, “Editing the Filter,” on page 390](#)
- ♦ [Section 6.1.3, “Testing Filters,” on page 394](#)
- ♦ [Section 6.1.4, “Viewing the Filter XML Source,” on page 400](#)
- ♦ [Section 6.1.5, “Additional Filter Options,” on page 406](#)

6.1.1 Accessing the Filter Editor

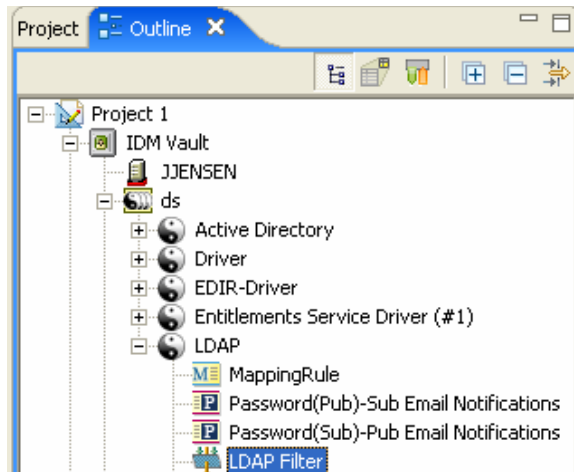
The Filter editor allows you to edit the filter. There are three different ways to access the Filter editor: through the model outline, through the policy flow, and through the Policy Set view.

Model Outline View


- 1 In an open project, click the *Outline* tab.
- 2 Click the *Show 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.

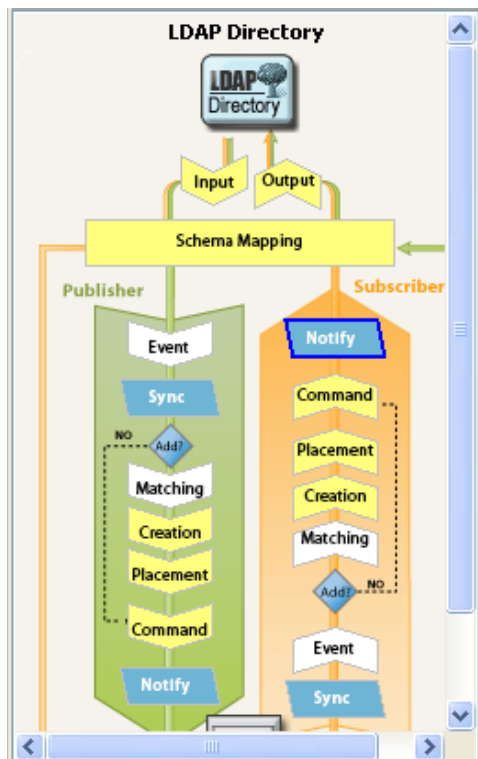
or

Right-click and select *Edit*.



Policy Flow View

- 1 In an open project, click the *Outline* tab.
- 2 Select the *Show Policy Flow* icon. 
- 3 Double-click the *Sync* icon or *Notify* icon to launch the Filter editor.

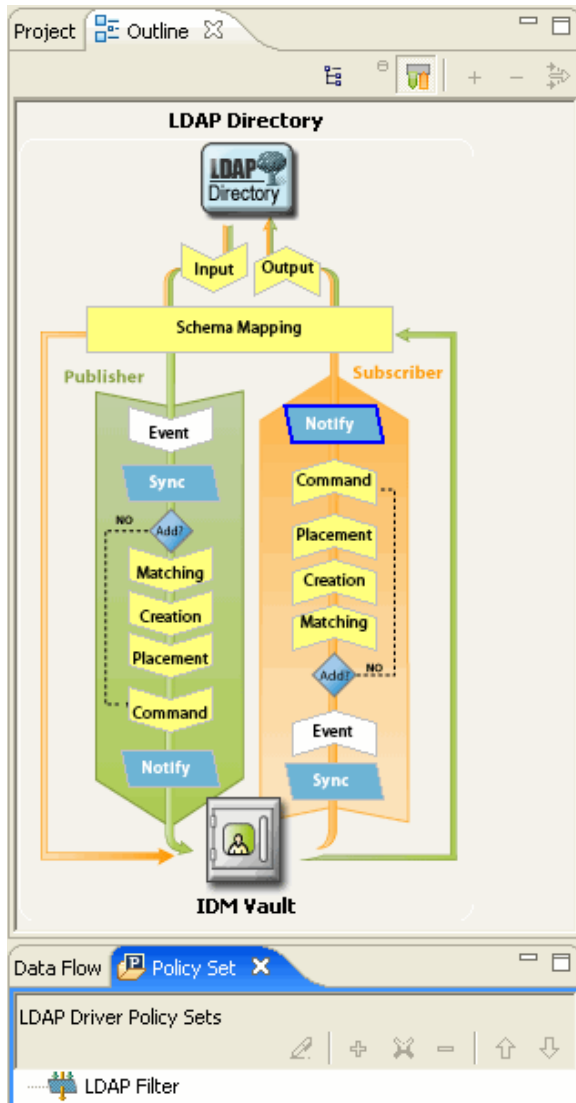


or

Right-click and select *Edit Policy > Filter*.

Policy Set View

- 1 Double-click the filter policy in the Policy Set view.



Keyboard Support

Table 6-1 Filter Editor Keyboard Support

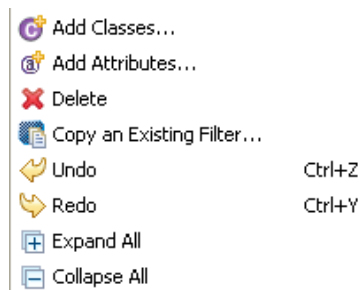
Action	Description
Up-arrow	Moves the cursor up in the Filter editor.
Down-arrow	Moves the cursor down in the Filter editor.
Left-arrow	Collapses the information displayed
Right-arrow	Expands the information displayed.
Insert	Adds a class.

Action	Description
Ctrl+Insert	Adds an attribute.
Delete	Deletes the selected items.
Enter	Accesses the edit mode. Press Enter a second time to commit the changes.
Esc	Exits the edit mode.

6.1.2 Editing the Filter

The Filter editor allows you to create and edit the filter. To display a context menu, right-click an item.

Figure 6-1 Filter Options




- ◆ “Removing or Adding Classes and Attributes” on page 390
- ◆ “Modifying Multiple Attributes” on page 391
- ◆ “Copying an Existing Filter” on page 391
- ◆ “Setting Default Values for Attributes” on page 391
- ◆ “Changing the Filter Settings” on page 392

Removing or Adding Classes and Attributes

By removing or adding classes and attributes, you determine the objects that 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 completely 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 Classes*.

or

Click the *Add Classes* 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*.

Modifying Multiple Attributes

The Filter editor allows you to modify more than one attribute at a time. Press the Ctrl key and select multiple attributes; when the option changes, it is changed for all of the selected attributes.

Copying an Existing Filter

You can copy an existing filter from another driver and use it in the driver you are currently working with.

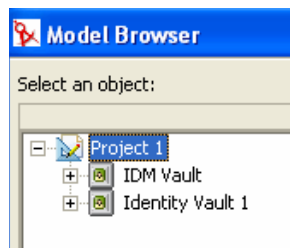
- 1 Click the *Copy an Existing Filter* icon 

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.



Setting Default Values for Attributes

You can define the default values for new attributes when they are added to the filter.

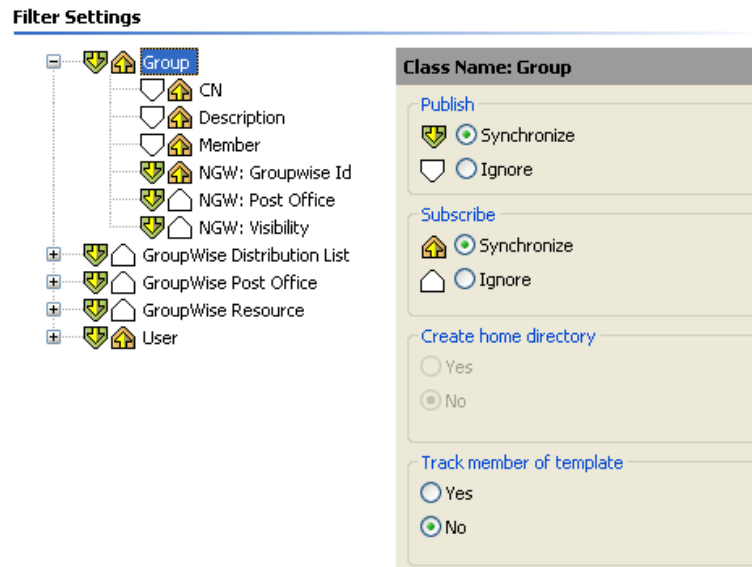
- 1 Click the *Set Default Values for New Attributes* icon  in the upper right corner.

- 2 Select the options you want new attributes to have, then click *OK*.

Changing the Filter Settings

The Filter editor gives you the option of changing how information is synchronized between the Identity Vault and the connected system. The filter has different settings for classes and attributes.

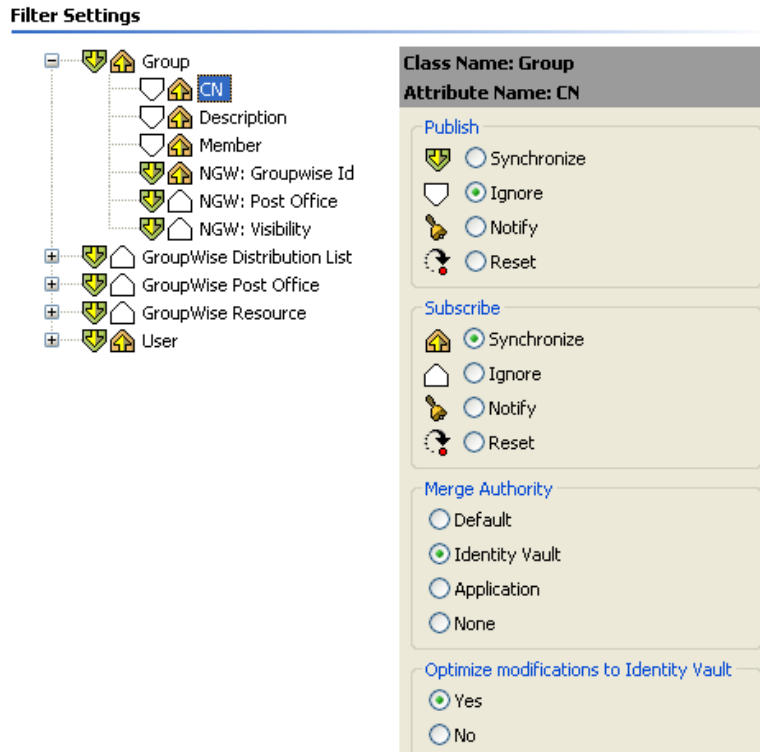
- 1 In the Filter editor, select a class.



- 2 Change the filter settings for the selected class.

Options	Definitions
<i>Publisher</i>	<ul style="list-style-type: none"> ◆ Synchronize: Allows the class to synchronize from the connected system into the Identity Vault. ◆ Ignore: Does not synchronize the class from the connected system into the Identity Vault.
<i>Subscriber</i>	<ul style="list-style-type: none"> ◆ Synchronize: Allows the class to synchronize from the Identity Vault into the connected system. ◆ Ignore: Does not synchronize the class from the Identity Vault into the connected system.
<i>Create Home Directory</i>	<ul style="list-style-type: none"> ◆ Yes: Automatically creates home directories. ◆ No: Does not create home directories.
<i>Track Member of Template</i>	<ul style="list-style-type: none"> ◆ 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.

3 Select an attribute.



4 Change the filter settings for the selected attribute.

Options	Definitions
<i>Publisher</i>	<ul style="list-style-type: none"> ◆ 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. ◆ Rest: Resets the object value to the value specified by the opposite channel. (You can set this value on either the Publisher channel or Subscriber channel, not both.)
<i>Subscriber</i>	<ul style="list-style-type: none"> ◆ 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 channel or Subscriber channel, not both.)

Options	Definitions
<i>Merge Authority</i>	<ul style="list-style-type: none"> ◆ Default Behavior: If an attribute is not being synchronized in either channel, no merging occurs. <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> ◆ 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. <p>This is valid behavior when synchronizing on the Subscriber channel.</p> ◆ Application: Behaves the same as the default behavior if the attribute is being synchronized on the Publisher channel and not on the Subscriber channel. <p>This is valid behavior when synchronizing on the Publisher channel.</p> ◆ <i>None:</i> No merging occurs regardless of synchronization.
<i>Optimize Modification to Identity Manager</i>	<ul style="list-style-type: none"> ◆ Yes: Changes to this attribute are examined on the Publisher channel to determine the minimal change made in the Identity Vault. ◆ No: Changes are not examined.

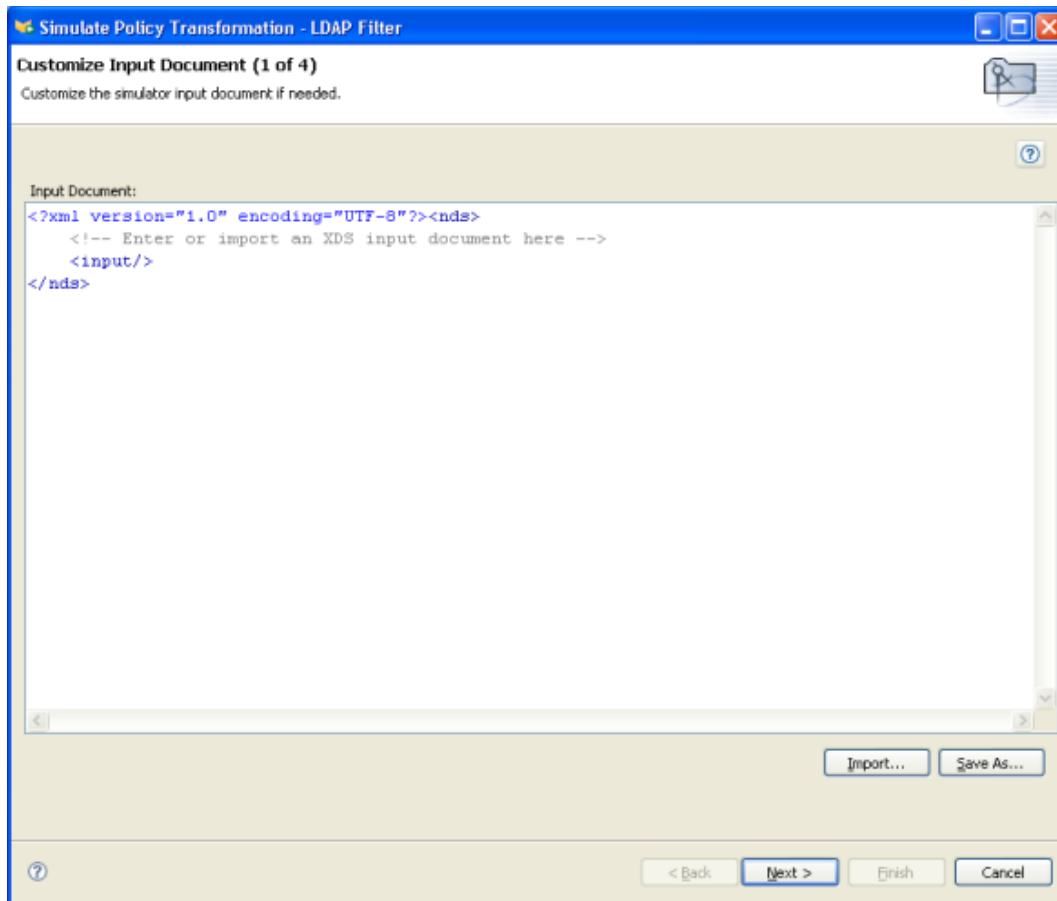
5 Click the *Save* icon  to save the changes.

6.1.3 Testing Filters

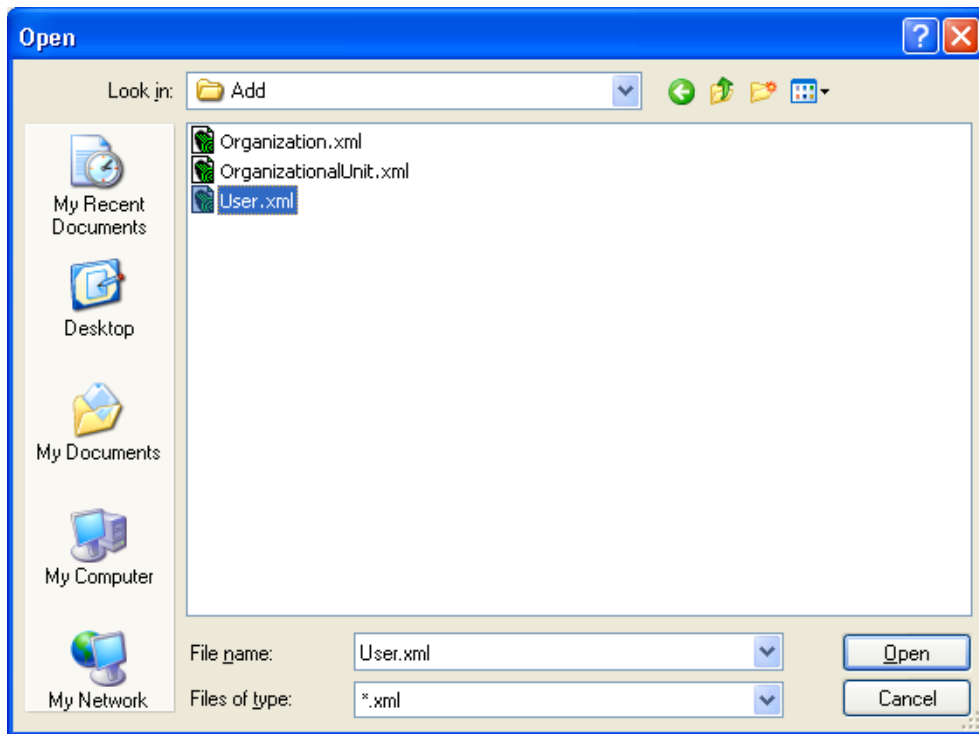
Designer comes with a tool called the Policy Simulator, which allows you to test your policies without implementing them in a production environment. You can launch the Policy Simulator through the Filter editor to test your policy after you have modified it.

1 Click the *Launch Policy Simulator* icon  in the toolbar.

2 Select *Import* to browse to a file that simulates an event.

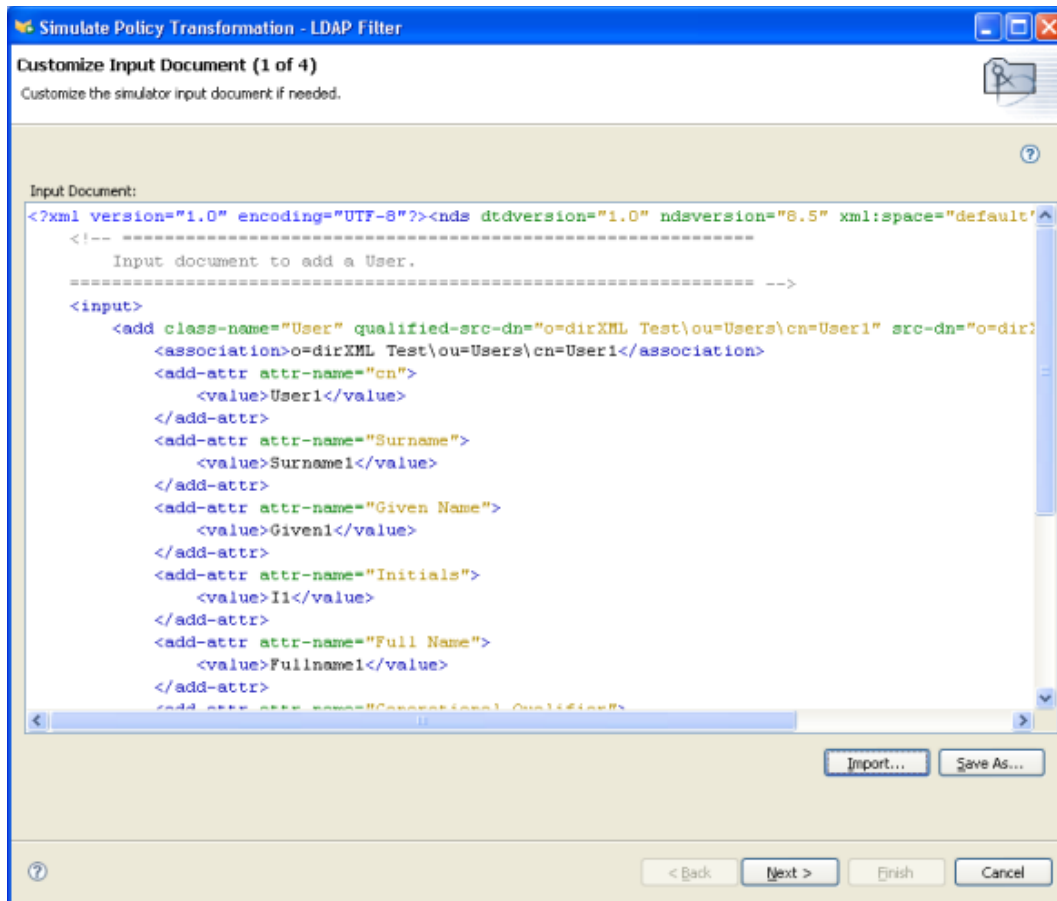


- 3 Select the file, then click *Open*. This example uses the `com.novell.designer.idm.policy\simulation\add\User.xml` file, which simulates an Add event for a User object.



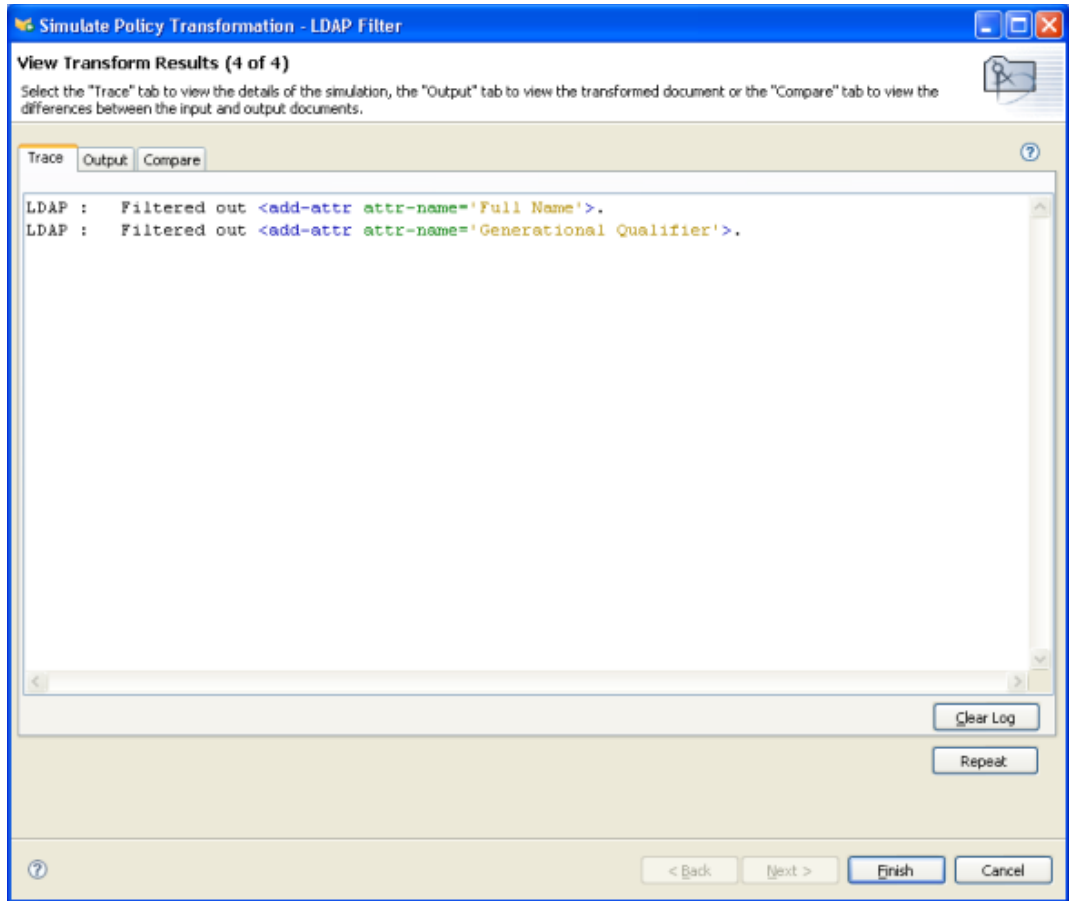
The Policy Simulator displays the input document of the user Add event.

- 4 Click *Next* to begin the simulation.

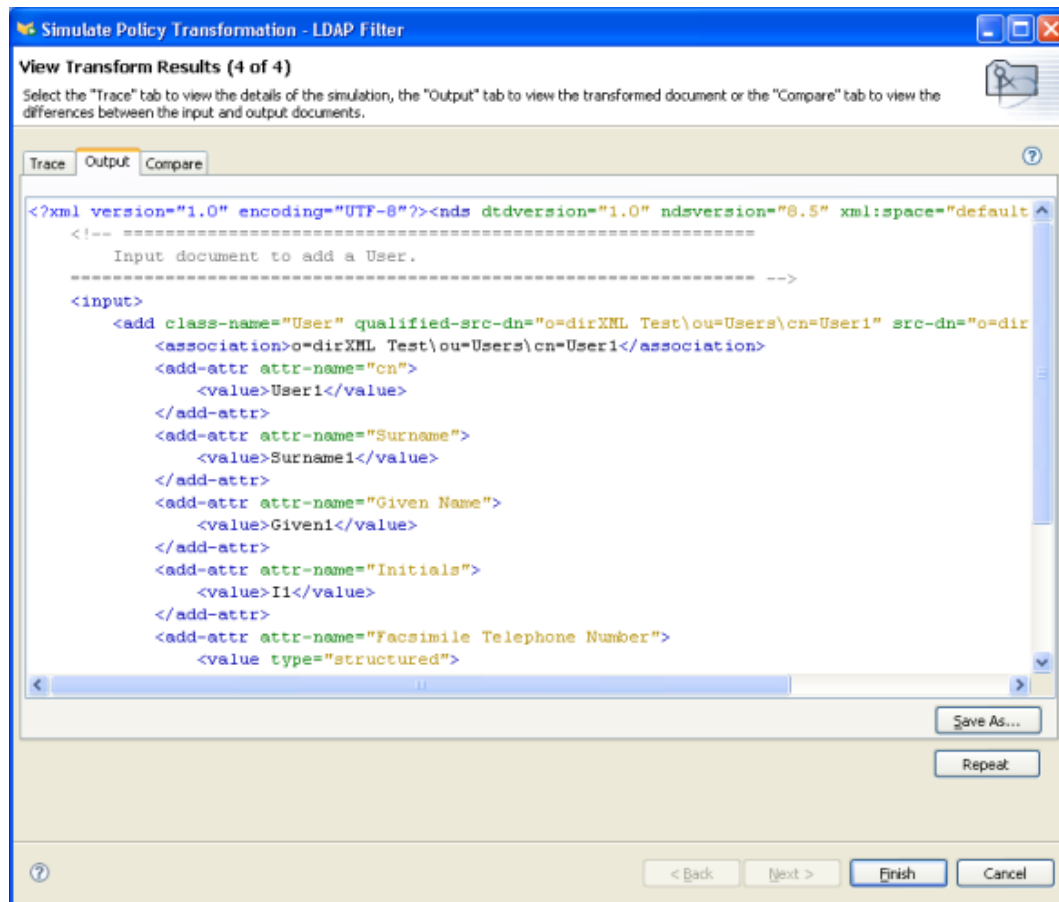


The Policy Simulator displays the log of the Add event, the output document, and a comparison of the Input document to the Output document that is generated.

- 5 Select the *Trace* tab display the results of the Add event as you see them in DSTRACE.

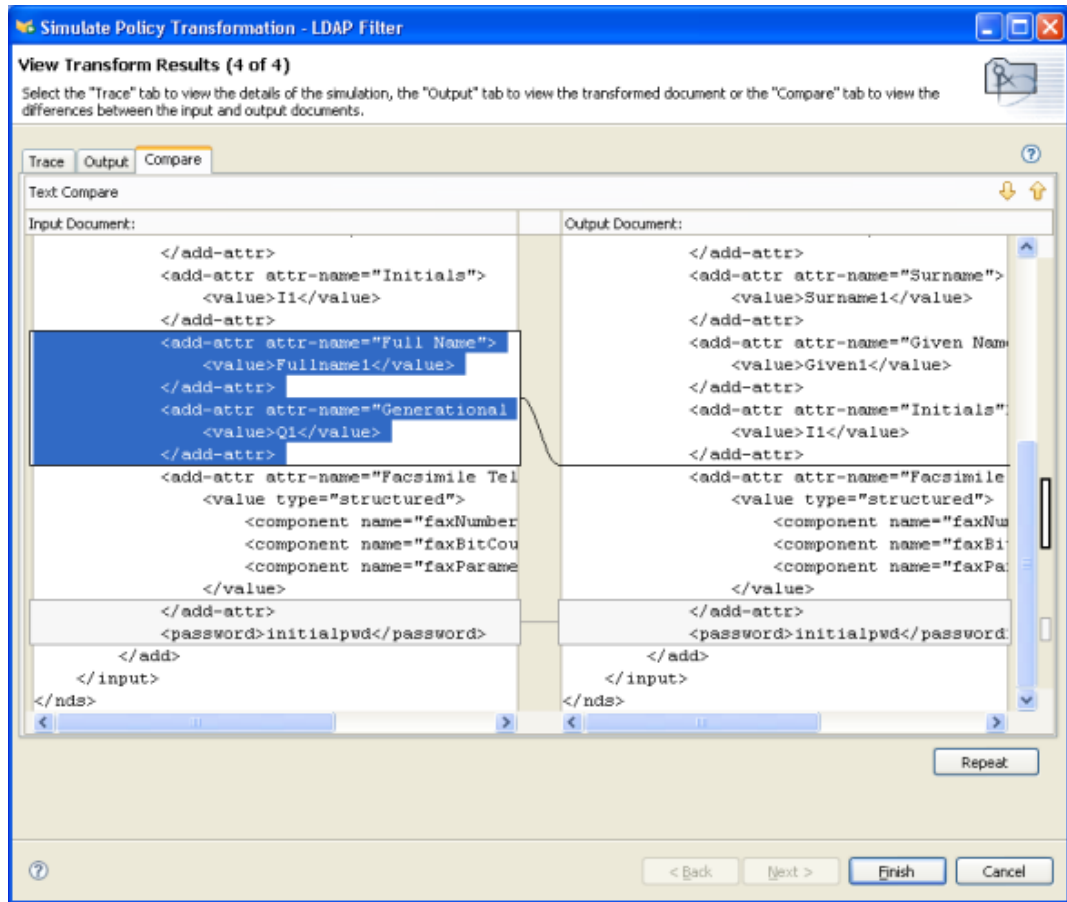


- 6 Select the *Output* tab to see the output document that is generated when the filter is executed against an input document. The input document is the user Add event.



You can edit the input and output documents. If you want to keep the changes, click *Save As*.

- 7 Select the *Compare* tab to compare the text of the input document to the output document that is generated.



- 8 Click *Repeat* to select a different input document and see the results of that event.
- 9 When you have finished testing the filter, click *Finish* to close the Policy Simulator.

6.1.4 Viewing the Filter XML Source

Designer enables you to view, edit, and validate the XML by using an XML editor or text editor.

- [“Viewing the XML Source” on page 400](#)
- [“Editing the XML Source” on page 403](#)
- [“Validating the XML Source” on page 406](#)

Viewing the XML Source

You can view the XML Source in XML or in the XML tree format.

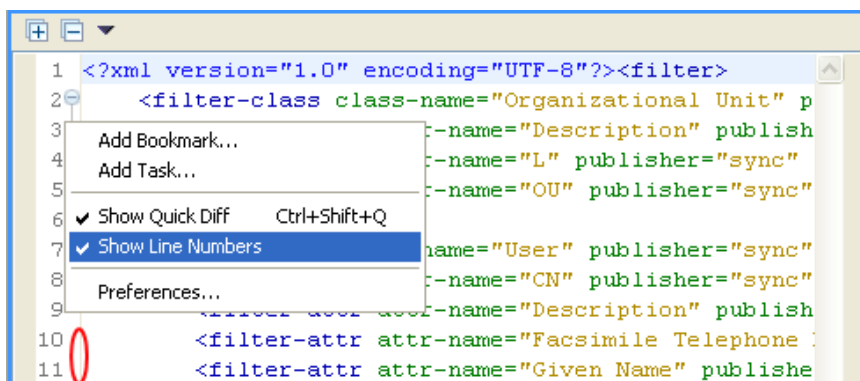
To open the XML Source view:

- 1 Click *XML Source* at the bottom of the Filter editor's workspace.



The XML editor displays line numbers. To see the line number, right-click in the left margin, then select *Show Line Numbers*.

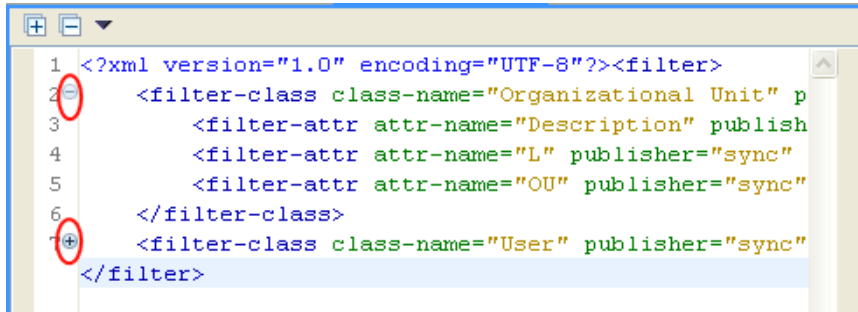
Figure 6-2 Filter Show Line Numbers



The XML editor expands or collapses the XML by function. If there are functions that contain a large amount of XML, you can collapse the XML by clicking the minus icon in the top left corner. To expand all of the XML functions, click the plus icon in the top left corner.

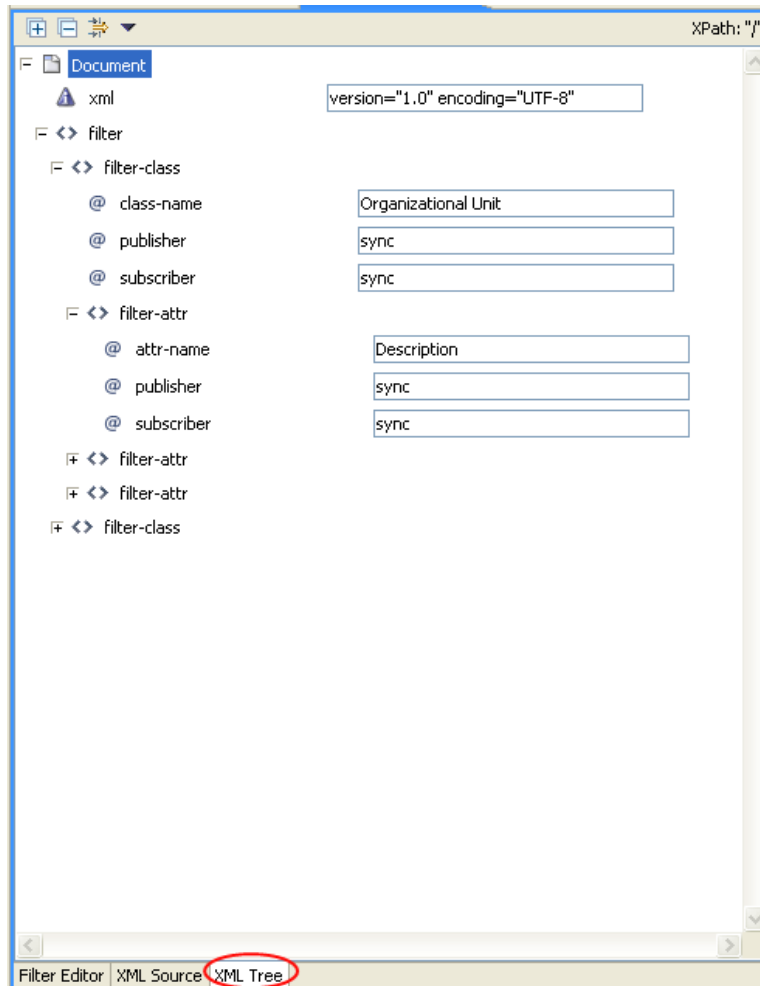
Each element has its own plus or minus icon in the left margin.

Figure 6-3 Filter XML Plus or Minus



To view the XML in the tree format:

- 1 Click *XML Tree* at the bottom of the Filter editor's workspace.

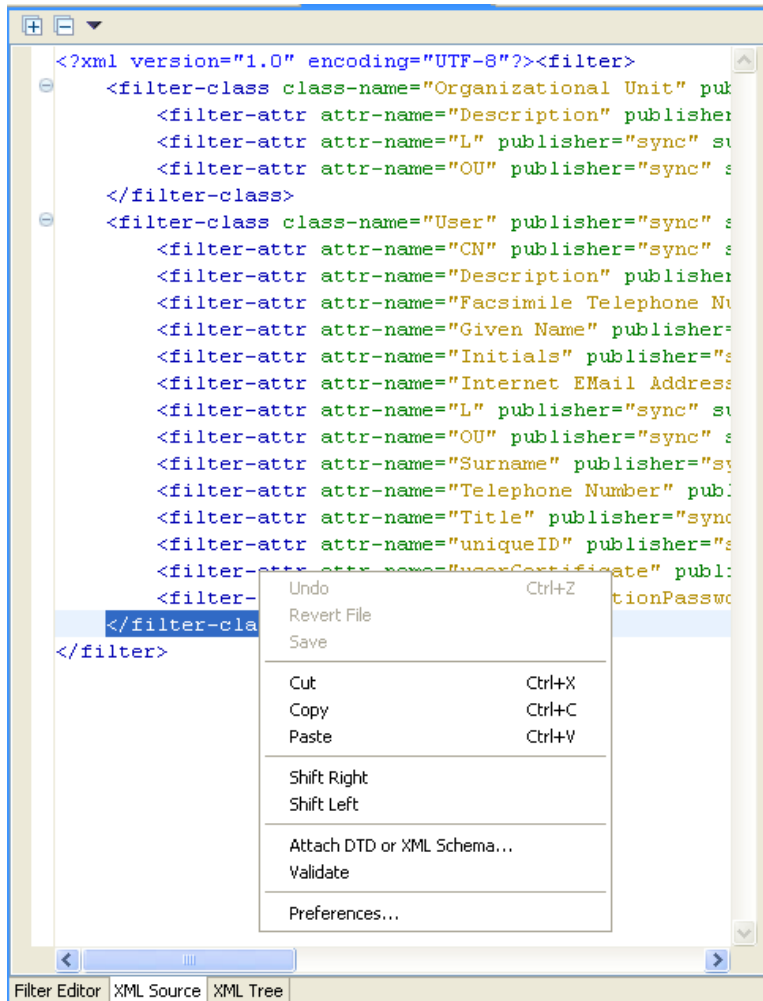


To see the entire tree view, expand each item listed.

Editing the XML Source

You can edit the XML through the XML editor. You can make changes here as well as through the GUI interface.

Figure 6-4 *Editing the XML Source of the Filter*



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 of the XML Source view is based on the editor that loads.

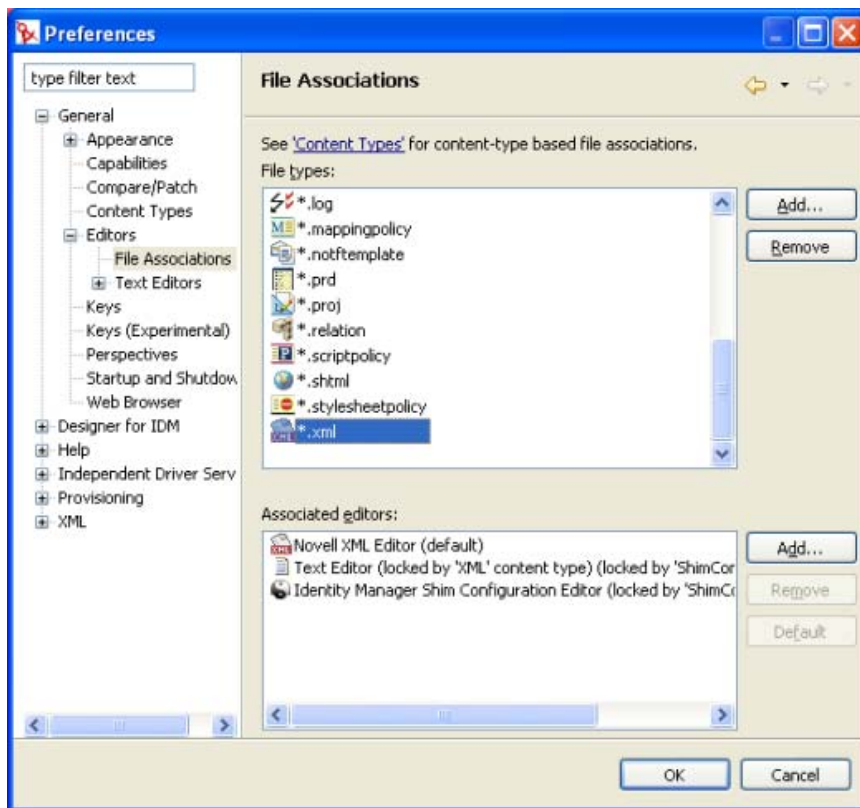
Right-click to display the list of the functions the XML editor contains.

- ◆ **Undo:** Undoes the last action.
- ◆ **Revert File:** Reverts the file to the last version that was saved.
- ◆ **Save:** Saves the file.
- ◆ **Cut:** Cuts the selected information.
- ◆ **Copy:** Copies the selected information to the Clipboard.

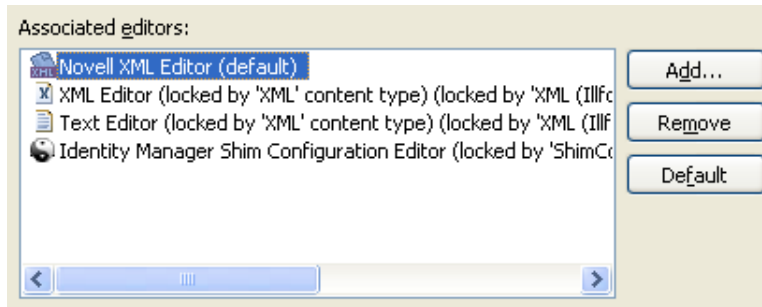
- ◆ **Paste:** Pastes the information into the document.
- ◆ **Shift Right:** Indents the line to the right.
- ◆ **Shift Left:** Indents the line to the left.
- ◆ **Attache DTD or XML Schema:** Attaches a DTD or XML schema file for validation of the policy.
- ◆ **Validate:** Validates the XML code.
- ◆ **Preferences:** Sets the preferences for the XML editor.

To choose a different XML editor for your XML 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*) from the Associated editors. (If the editor you want isn't in the list, you can click *Add*, then add it to the list.)

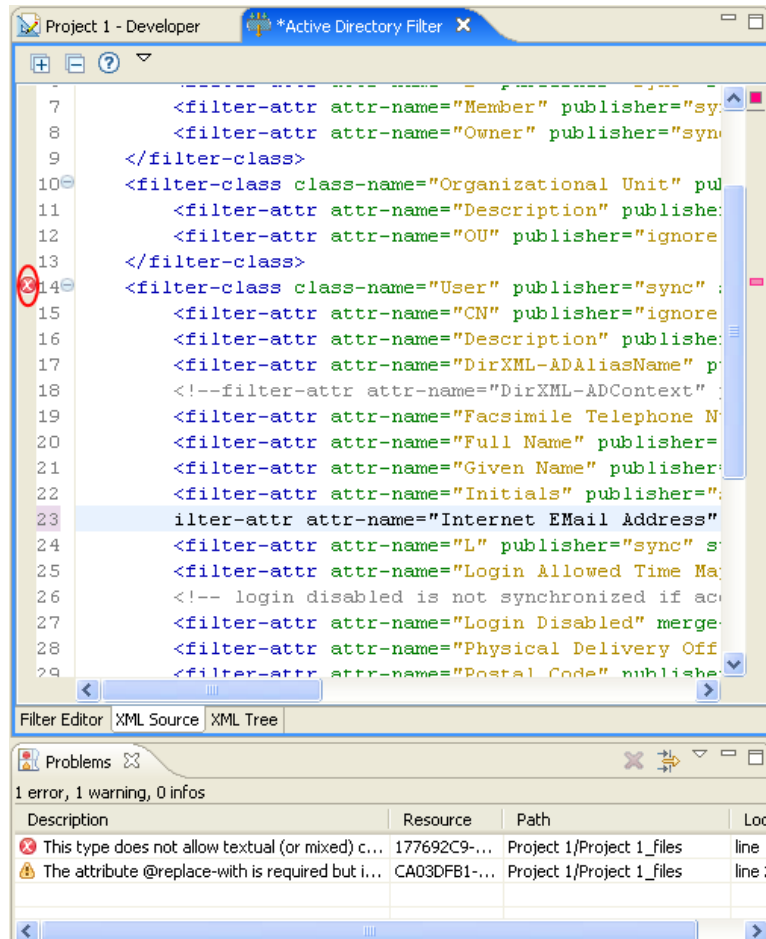


- 5 Click *OK*.
- 6 Close and reopen the Filter editor. The default editor should be loaded in the *XML Source* view.

Validating the XML Source

The XML editor validates the XML code. Right-click, then select *Validate*. If there are errors, a red x is displayed on the line where the error occurs. An explanation at the bottom of the window gives more information about the problem.

Figure 6-5 *Validate Filter*



In this example, the beginning tag and the first letter of the `<filter-attr>` are missing.

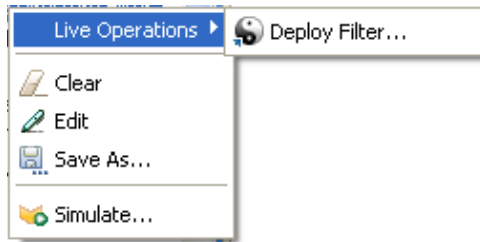
6.1.5 Additional Filter Options

When you right-click on a filter object, there are multiple options presented in the Outline view, the Policy Flow view, and the Policy Set view.

- ◆ [“Outline View Additional Options” on page 407](#)
- ◆ [“Policy Flow View Additional Options” on page 407](#)
- ◆ [“Policy Set View Additional Options” on page 407](#)

Outline View Additional Options

- 1 Right-click the filter object in the Outline view.



- ♦ **Live Operations > Deploy Filter:** Deploys the filter into the Identity Vault.
- ♦ **Clear:** Deletes all content from the filter policy, but leaves the object.
- ♦ **Edit:** Launches the Filter editor. For more information, see [Section 6.1.2, “Editing the Filter,” on page 390](#).
- ♦ **Save As:** Saves the Filter as a .xml file.
- ♦ **Simulate:** Launches the Policy Simulator. For more information, see [Section 6.1.3, “Testing Filters,” on page 394](#).

Policy Flow View Additional Options

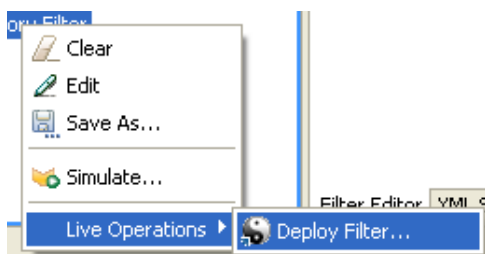
- 1 Right-click the filter object in the Policy Flow view.



- ♦ **Edit Policy > Filter:** Launches the Filter editor. For more information, see [Section 6.1.2, “Editing the Filter,” on page 390](#).
- ♦ **Simulate:** Launches the Policy Simulator. For more information, see [Section 6.1.3, “Testing Filters,” on page 394](#).

Policy Set View Additional Options

- 1 Right-click the filter object in the Policy Set view.



- ♦ **Clear:** Deletes all content from the filter policy, but leaves the object.
- ♦ **Edit:** Launches the Filter editor. For more information, see [Section 6.2.2, “Editing the Filter,” on page 408](#).

- ♦ **Save:** Saves the filter as a Xml file.
- ♦ **Simulate:** Launches the Policy Simulator. For more information, see [Section 6.1.3, “Testing Filters,” on page 394.](#)
- ♦ **Live Operations > Deploy Filter:** Allows you to deploy the filter into the Identity Vault.

6.2 Filter Tasks in iManager

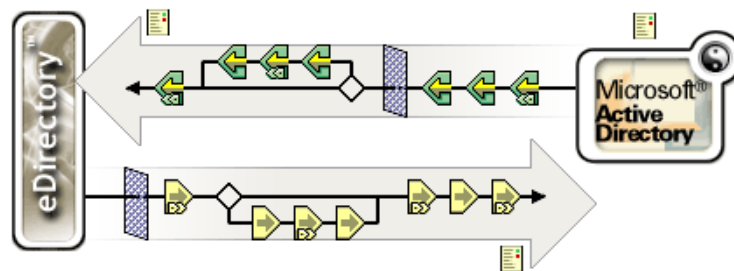
This section contains instructions on performing common filter-related tasks in iManager:

- ♦ [Section 6.2.1, “Accessing the Filter,” on page 408](#)
- ♦ [Section 6.2.2, “Editing the Filter,” on page 408](#)

6.2.1 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 6-6 Driver Overview



- 4 Click the Filter icon on the Publisher or Subscriber channel. It is the same object.



6.2.2 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 409](#)
- ♦ [“Adding a Class” on page 409](#)
- ♦ [“Adding an Attribute” on page 409](#)
- ♦ [“Copying a Filter” on page 409](#)
- ♦ [“Setting a Template” on page 409](#)
- ♦ [“Changing the Filter Settings” on page 409](#)

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.

Changing the Filter Settings

The Filter editor gives you the option of changing how information is synchronized between the Identity Vault and the connected system. The filter has different settings for classes and attributes.

- 1 In the Filter editor, select a class.
- 2 Change the filter settings for the selected class.

Options	Definitions
<i>Publisher</i>	<ul style="list-style-type: none">◆ Synchronize: Allows the class to synchronize from the connected system into the Identity Vault.◆ Ignore: Does not synchronize the class from the connected system into the Identity Vault.
<i>Subscriber</i>	<ul style="list-style-type: none">◆ Synchronize: Allows the class to synchronize from the Identity Vault into the connected system.◆ Ignore: Does not synchronize the class from the Identity Vault into the connected system.

Options	Definitions
<i>Create Home Directory</i>	<ul style="list-style-type: none"> ◆ Yes: Automatically creates home directories. ◆ No: Does not create home directories.
<i>Track Member of Template</i>	<ul style="list-style-type: none"> ◆ 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.

3 Select an attribute.

4 Change the filter settings for the selected attribute.

Options	Definitions
<i>Publisher</i>	<ul style="list-style-type: none"> ◆ 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. ◆ Rest: Resets the object value to the value specified by the opposite channel. (You can set this value on either the Publisher channel or Subscriber channel, not both.)
<i>Subscriber</i>	<ul style="list-style-type: none"> ◆ 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 channel or Subscriber channel, not both.)

Options	Definitions
<i>Merge Authority</i>	<ul style="list-style-type: none"> <li data-bbox="558 260 1281 506">♦ Default Behavior: If an attribute is not being synchronized in either channel, no merging occurs. 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. 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). 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. This is always valid behavior. <li data-bbox="558 873 1281 1031">♦ 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. This is valid behavior when synchronizing on the Subscriber channel. <li data-bbox="558 1041 1281 1199">♦ Application: Behaves the same as the default behavior if the attribute is being synchronized on the Publisher channel and not on the Subscriber channel. This is valid behavior when synchronizing on the Publisher channel. <li data-bbox="558 1209 1182 1241">♦ <i>None:</i> No merging occurs regardless of synchronization.
<i>Optimize Modification to Identity Manager</i>	<ul style="list-style-type: none"> <li data-bbox="558 1262 1240 1346">♦ Yes: Changes to this attribute are examined on the Publisher channel to determine the minimal change made in the Identity Vault. <li data-bbox="558 1356 927 1396">♦ No: Changes are not examined.

5 Click *OK* to save the changes.

Managing Schema Mapping Policies

7

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 7.1, “Schema Mapping Policy Tasks in Designer,” on page 413](#)
- ♦ [Section 7.2, “Schema Mapping Policy Tasks in iManager,” on page 436](#)

7.1 Schema Mapping Policy Tasks in Designer

This section contains instructions on performing common tasks related to Schema Mapping policies in Designer:


- ♦ [Section 7.1.1, “Accessing the Schema Map Editor,” on page 413](#)
- ♦ [Section 7.1.2, “Editing a Schema Mapping Policy,” on page 417](#)
- ♦ [Section 7.1.3, “Testing Schema Mapping Policies,” on page 420](#)
- ♦ [Section 7.1.4, “Accessing the Schema Mapping Policy XML,” on page 426](#)
- ♦ [Section 7.1.5, “Additional Schema Map Policy Options,” on page 432](#)

7.1.1 Accessing the Schema Map Editor

The Schema Map editor allows you to edit the Schema Mapping policies. There are three different ways to access the Schema Map editor in Designer: through the Outline view, through the Policy Flow view, or through the Policy Set view.

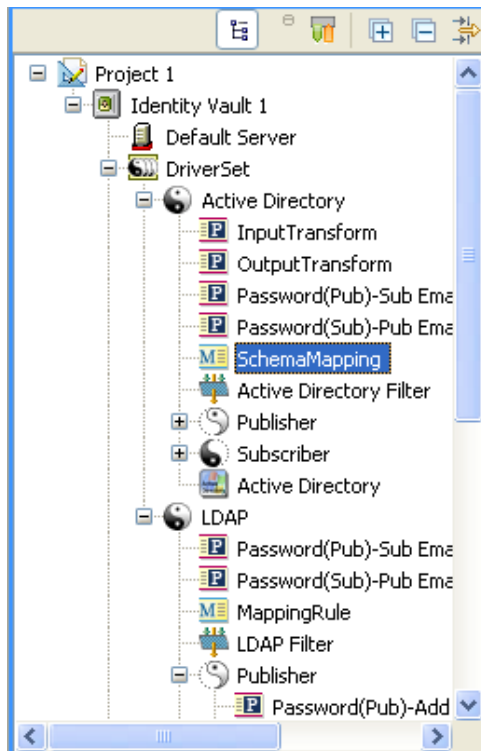
- ♦ [“Outline View” on page 413](#)
- ♦ [“Policy Flow View” on page 414](#)
- ♦ [“Policy Set View” on page 415](#)
- ♦ [“KeyBoard Support” on page 416](#)

Outline View


- 1 In an open project, click the *Outline* tab.
- 2 Click the *Show 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.

or

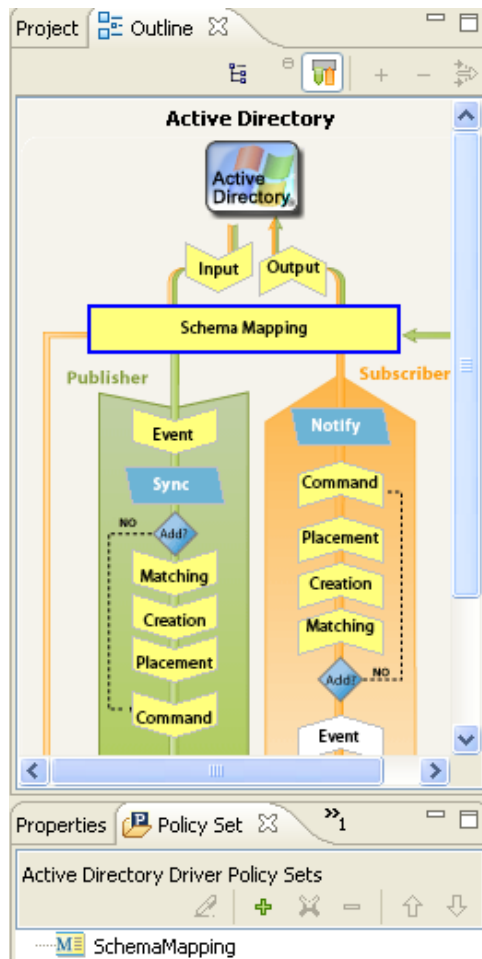
Right-click and select *Edit*.



Policy Flow View

- 1 In an open project, click the *Outline* tab.
 - 2 Click the *Show Policy Flow* icon. 
 - 3 Double-click the Schema Mapping policy to launch the Schema Map editor.
- or

Right-click and select *Edit Policy* to launch the Schema Map editor.

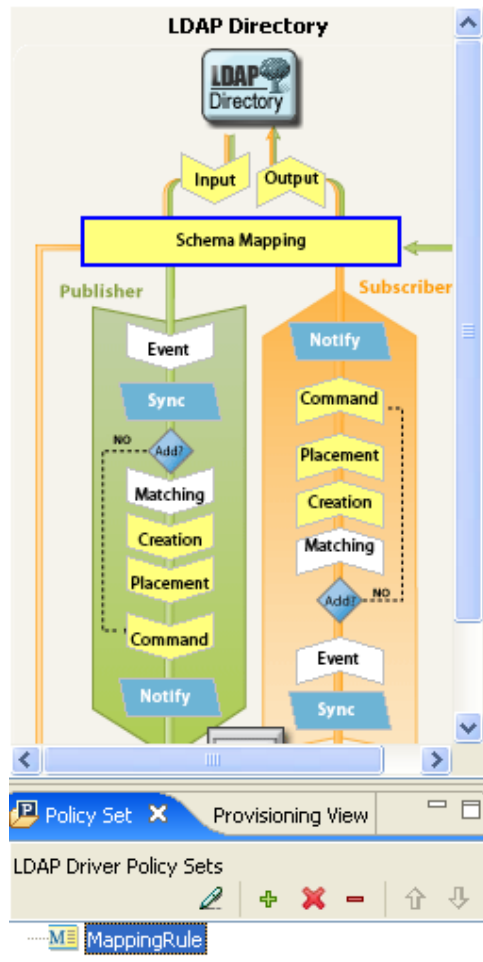


Policy Set View

1 Double-click the Schema Map policy in the Policy Set view.

or

Right-click the Schema Map policy and select *Edit*.



KeyBoard Support

Table 7-1 Schema Map Editor Keyboard Support

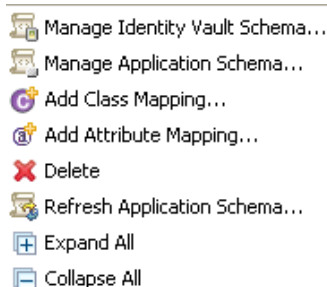
Action	Description
Up-arrow	Moves the cursor up in the Schema Map editor.
Down-arrow	Moves the cursor down in the Schema Map editor.
Left-arrow	Collapses the information displayed
Right-arrow	Expands the information displayed.
Insert	Adds a class.
Ctrl+Insert	Adds an attribute.
Delete	Deletes the selected items.
Enter	Accesses the edit mode. Press Enter a second time to commit the changes.

Action	Description
Esc	Exits the edit mode.

7.1.2 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.

Figure 7-1 Context Menu of the Schema Map Editor




- ◆ [“Removing or Adding Classes and Attributes” on page 417](#)
- ◆ [“Refreshing the Application Schema” on page 418](#)
- ◆ [“Editing Items” on page 419](#)
- ◆ [“Sorting Items” on page 419](#)
- ◆ [“Managing Schema” on page 419](#)

Removing or Adding Classes and Attributes

- ◆ [“Removing a Class or Attribute” on page 417](#)
- ◆ [“Adding a Class” on page 418](#)
- ◆ [“Adding a Attribute” on page 418](#)

Removing a Class or Attribute

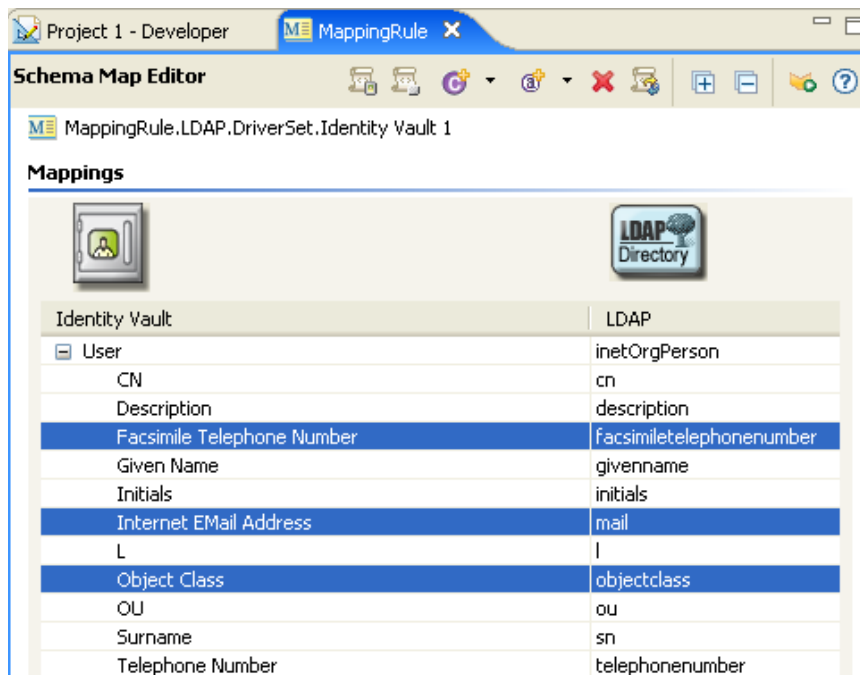
If you do not want a class or an attribute to be mapped to a class or attribute in the connected system, the best practice is to completely remove the class or the attribute from the Schema Mapping policy. There are three different ways to add or remove attributes and classes from the Schema Mapping policy:

- ◆ Select the class or attribute you want to remove, then right-click 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 to delete the items.



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*.

or

Select the *Add Attribute Mapping* icon  in the upper-right corner.

- 2 From the drop-down list for the Identity Vault, select the attribute you want to add.
- 3 From the drop-down list for the connected system, select the attribute you want to add.
- 4 To save the changes, click *File > Save*.

Refreshing the Application Schema

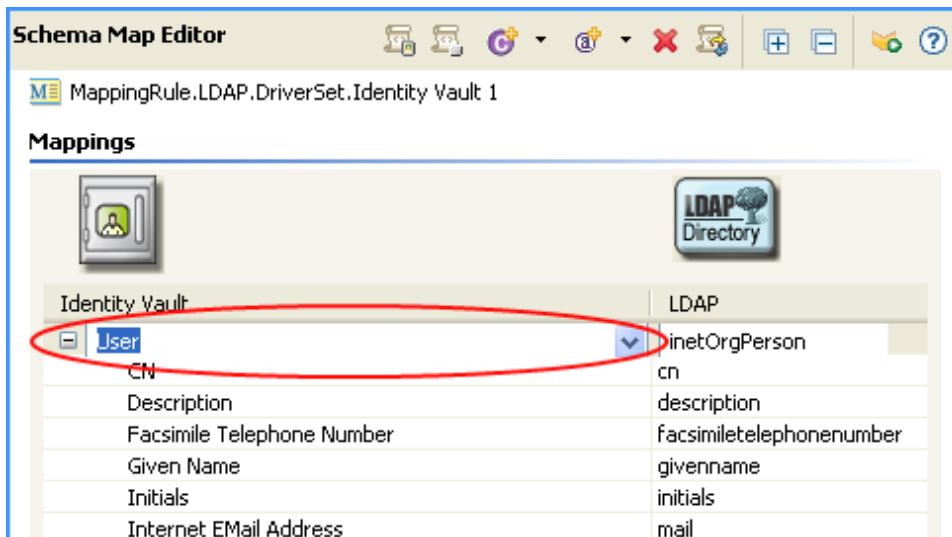
If you have modified the schema in the application, these changes need to be reflected in the Schema Mapping policy. To make the new schema available, click the *Refresh application schema* icon  in the toolbar.

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.

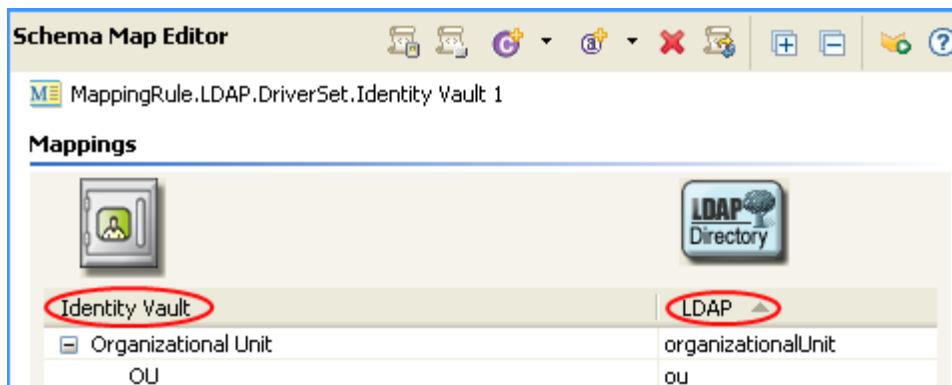
Figure 7-2 Schema Map Editor



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.

Figure 7-3 Schema Map Editor Sorting Items



Managing Schema


Designer allows you to manage the Identity Vault schema and any connected system's schema. You can import the schema, modify it, and deploy the changed schema back into the Identity Vault or the connected systems. To manage the Identity Vault schema, right-click in the Schema Map editor and click *Manage Identity Vault Schema*. To manage the connected systems schema, right-click in the

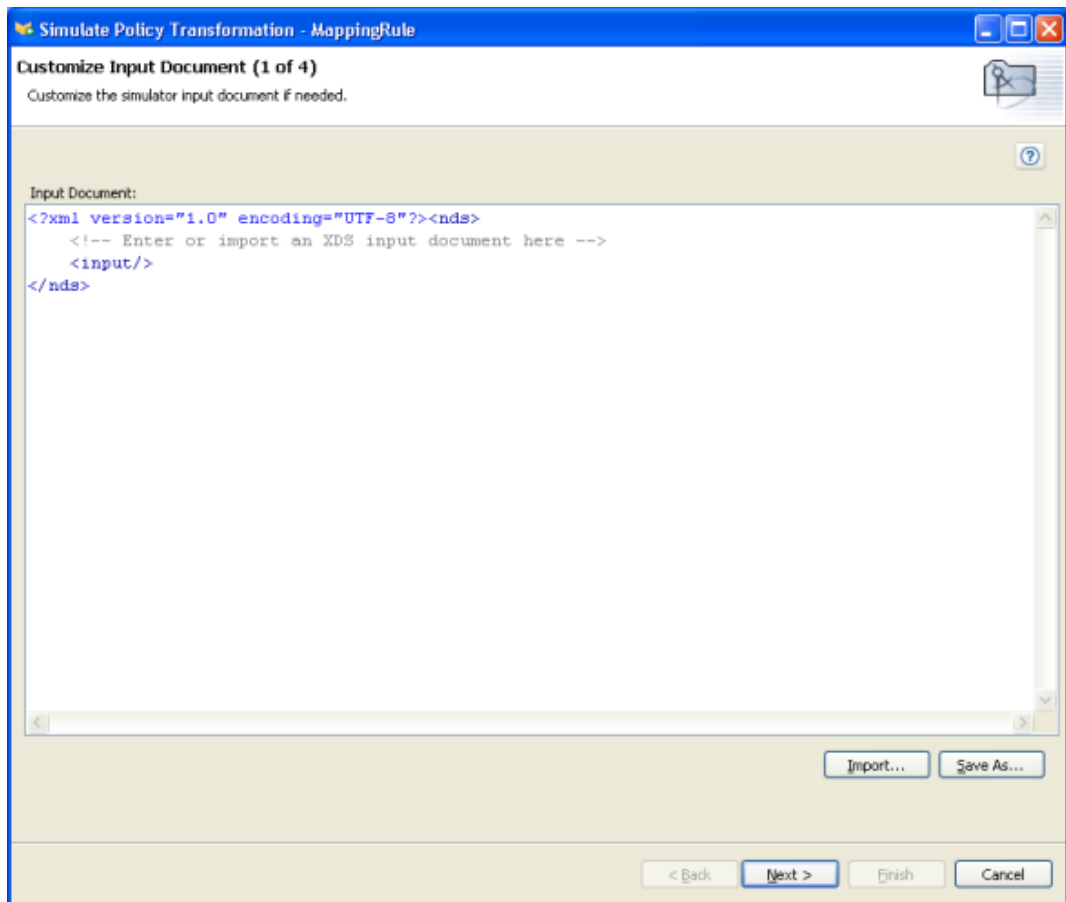
Schema Map editor and click *Manage Application Schema*. For information about how to manage the schema, see “[Managing the Schema](#)” in the *Designer for Identity Manager 3: Administration Guide*.

7.1.3 Testing Schema Mapping Policies

Designer comes with a tool called the Policy Simulator. It allows you to test your policies without implementing them a production environment. You can launch the Policy Simulator through the Schema Mapping editor to test your policy after you have modified it.

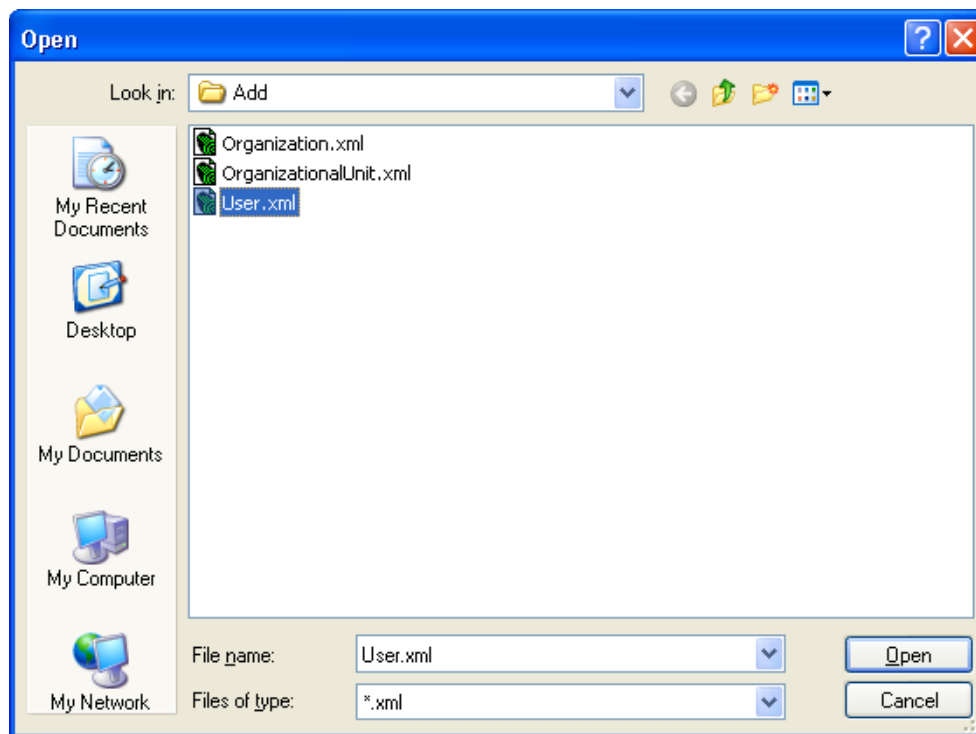
To access the Policy Simulator and test the Schema Mapping policy:

- 1 Click the *Launch Policy Simulator* icon  in the toolbar.
- 2 Select *Import* to browse to a file that simulates an event.



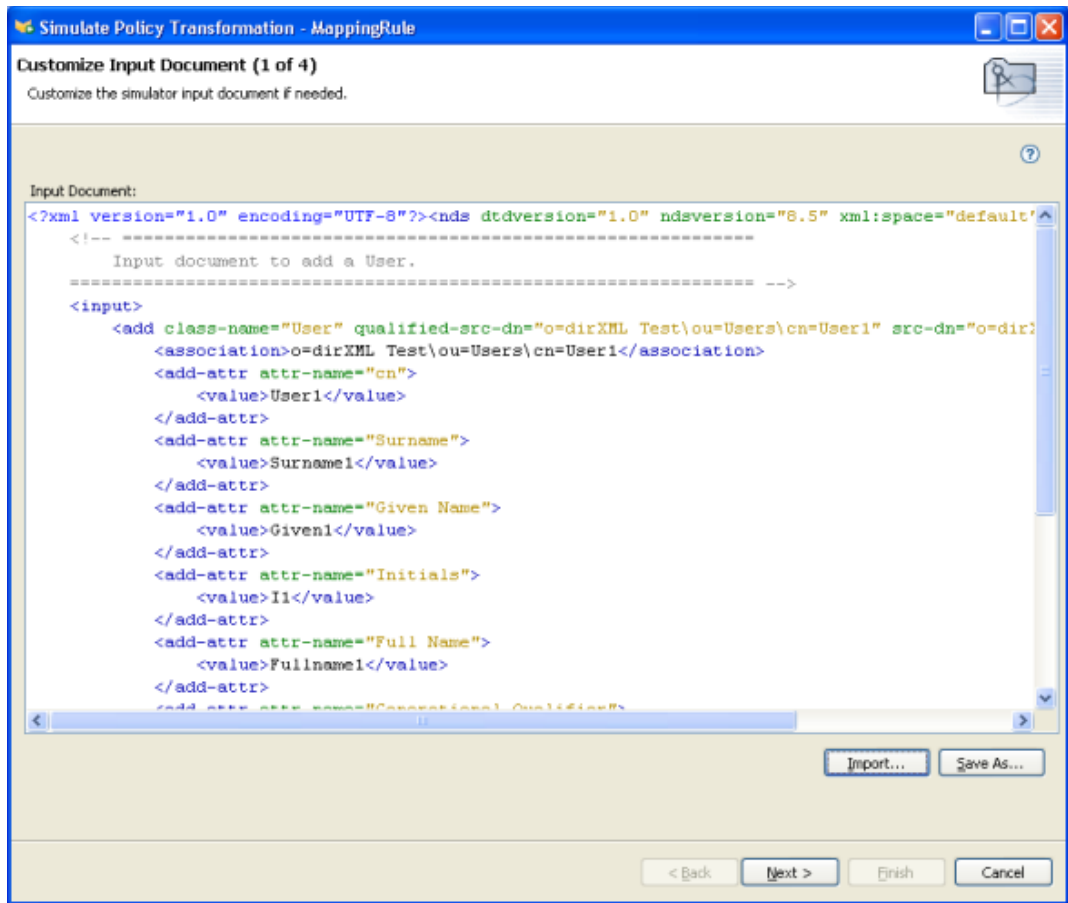
- 3 Select the file, then click *Open*.

This example uses the `com.novell.designer.policy\simulation\add\user.xml` file, which simulates an Add event of a user object.

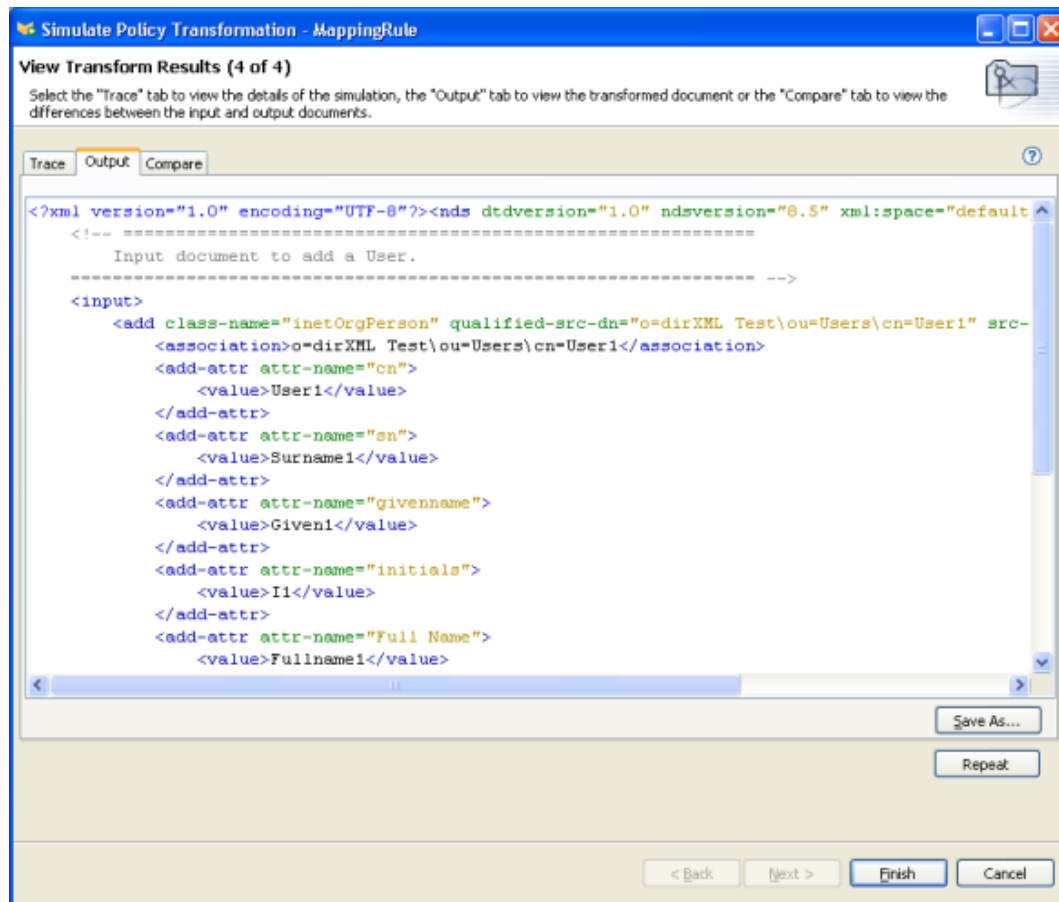


The Policy Simulator displays the input document of the user Add event.

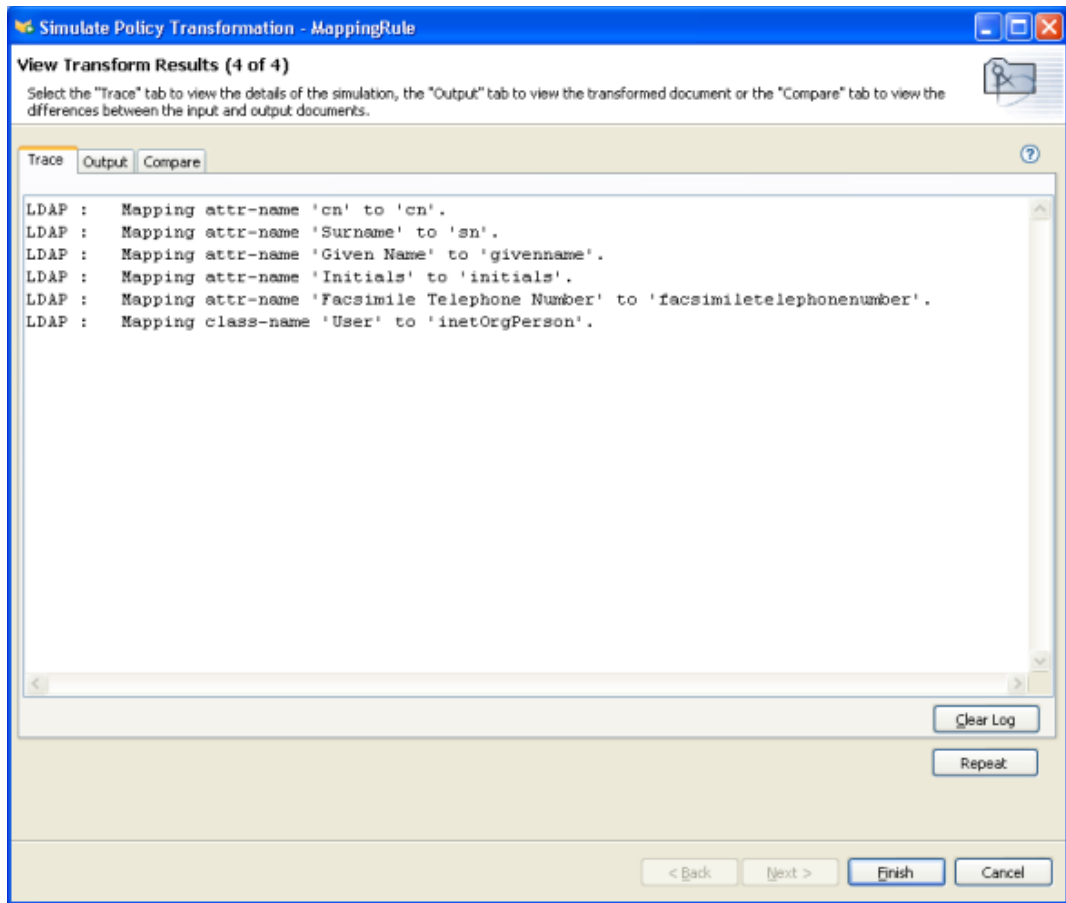
4 Click *Next* to begin the simulation.



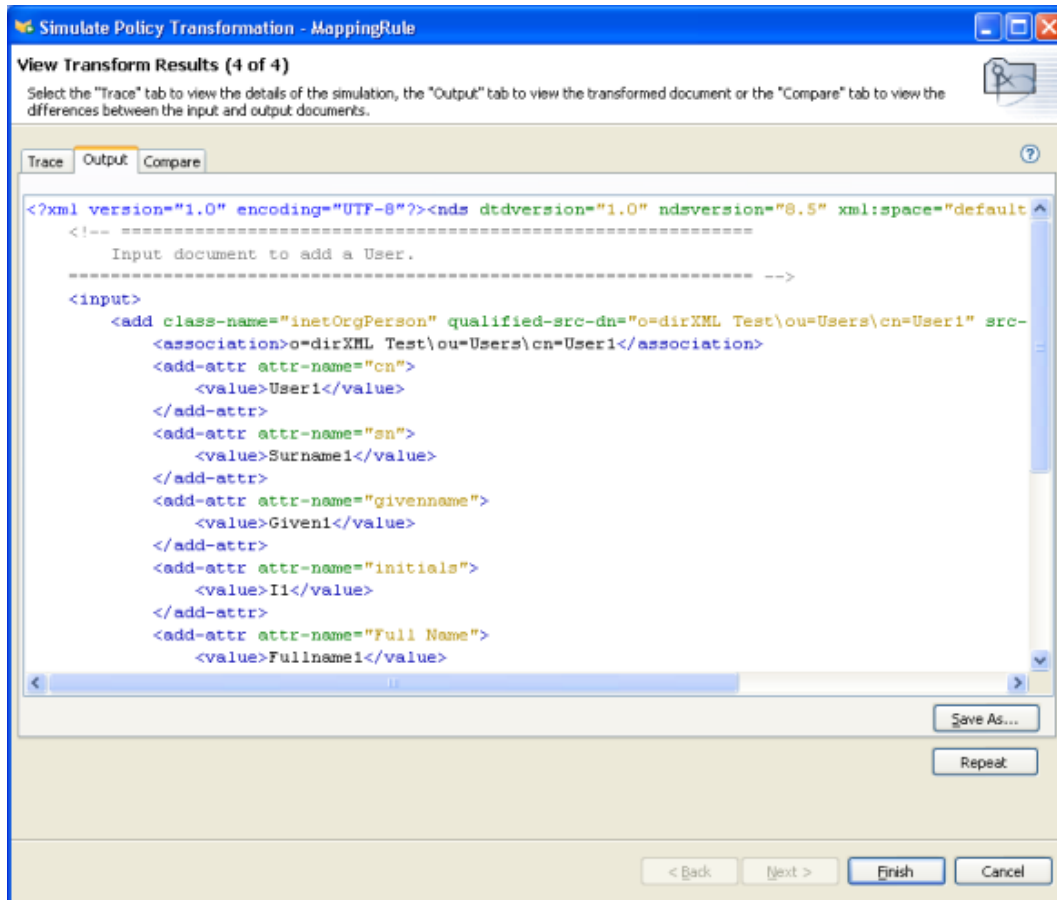
The Policy Simulator displays the log of the Add event, the output document, and a comparison of the input document to the output document that was generated.



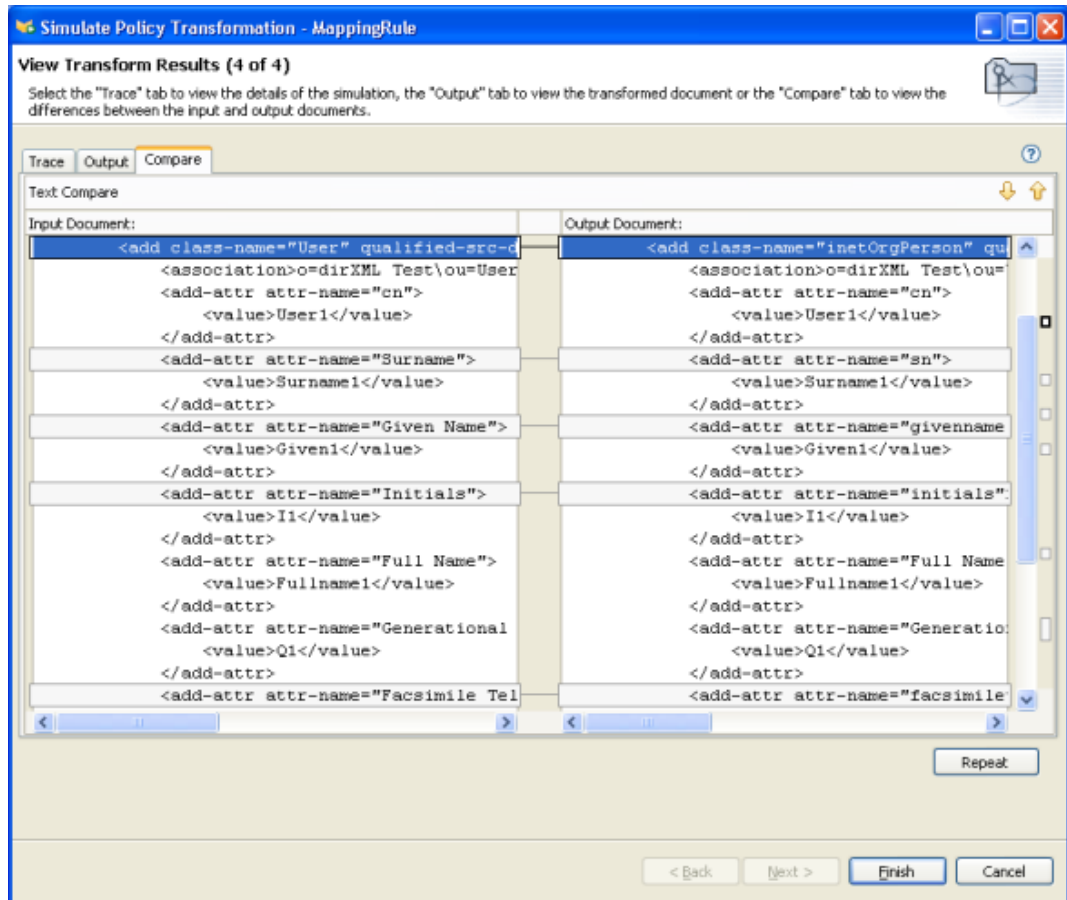
- 5 Select the *Trace* tab to see the results of the Add event as you would through DSTRACE.



- 6 Select the *Output* tab to view the output document that is generated from the Schema Map policy executed against the input document. In this example, it is the user Add event.



- 7 Select the *Compare* tab to compare the text of the input document to the document that is generated, which is the output document.



- 8 Click *Repeat* to select a different input document and see the results of that event.
- 9 When you have finished testing the Schema Mapping Policy, click *Finish* to close the Policy Simulator.

7.1.4 Accessing the Schema Mapping Policy XML

Designer enables you to view, edit, and validate the XML by using an XML editor or text editor.

- ◆ [“Viewing the XML Source” on page 426](#)
- ◆ [“Editing the XML Source” on page 430](#)
- ◆ [“Validating the XML Source” on page 432](#)

Viewing the XML Source

You can view the XML Source in XML or in the XML tree format.

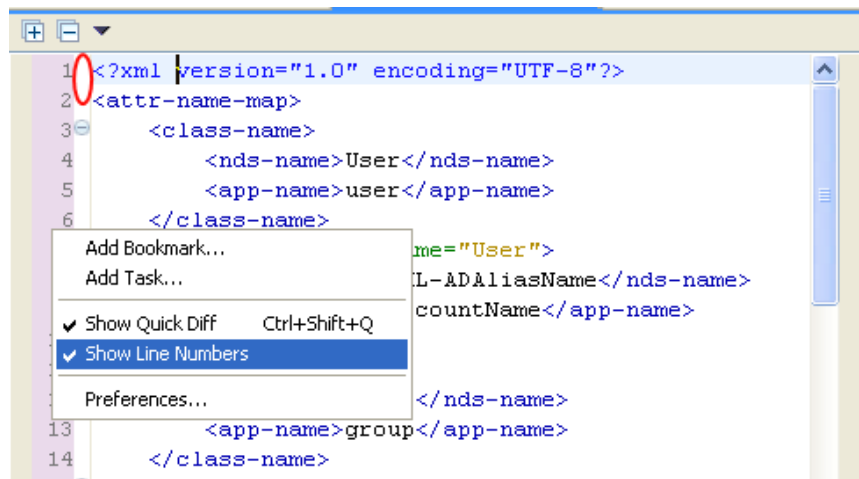
To open the XML Source view:

- 1 Click *XML Source* at the bottom of the Schema Map editor's workspace.



The XML editor displays line numbers. To see the line number, right-click in the left margin, then select Show Line Numbers.

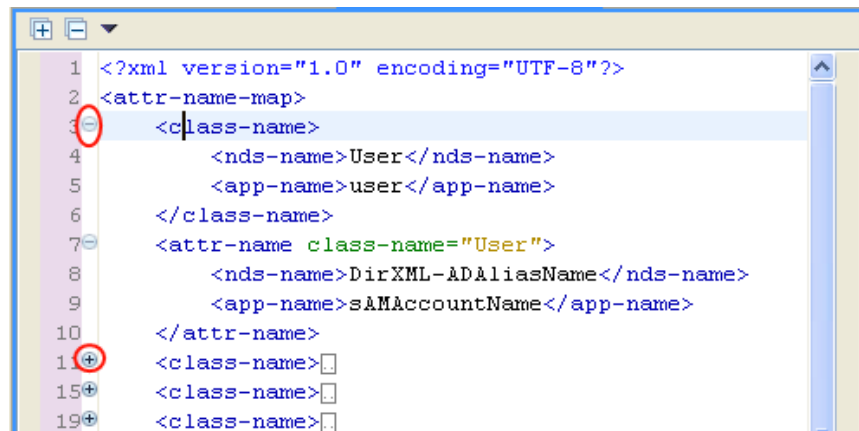
Figure 7-4 Schema Map Policy Line Numbers



The XML editor expands or collapses the XML by function. If there are functions that contain a large amount of XML, you can collapse the XML by clicking the minus icon in the top left corner. To expand all of the XML functions, click the plus icon in the top left corner.

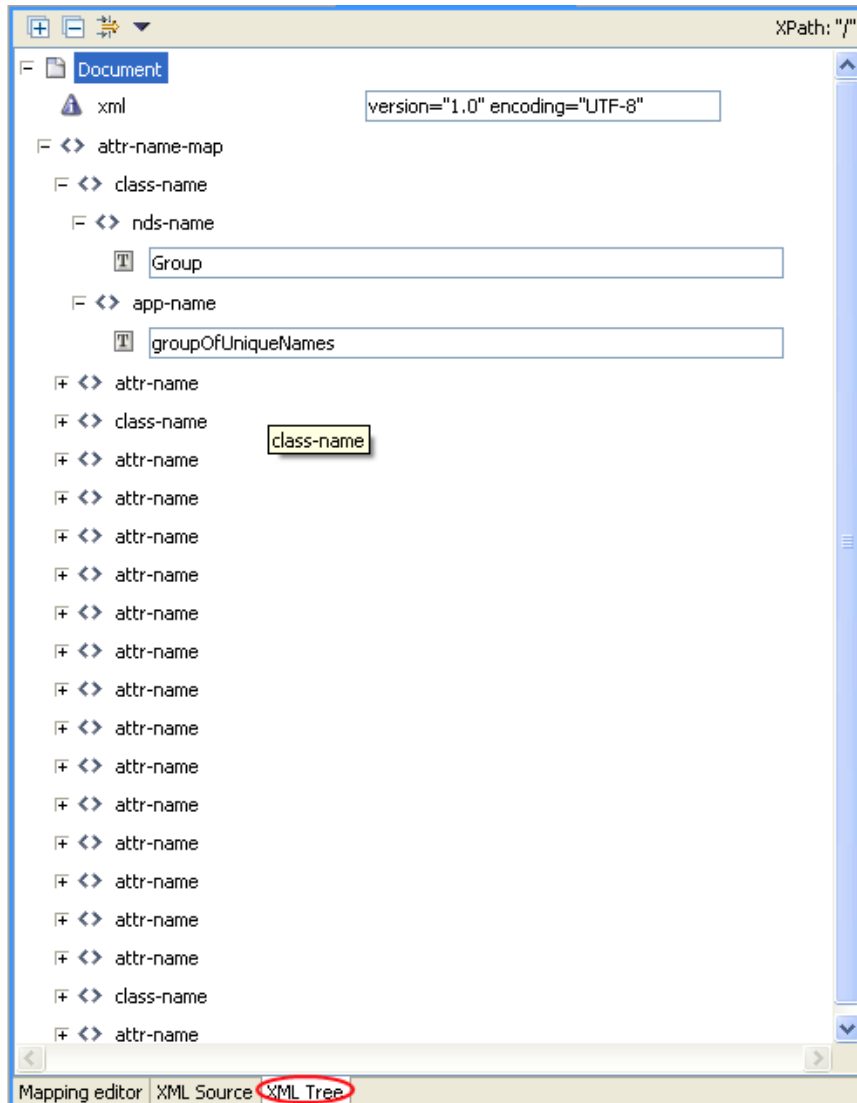
Each element has its own plus or minus icon in the left margin.

Figure 7-5 Schema Map Policy XML Plus or Minus



To view the XML in the tree format:

- 1 Click *XML Tree* at the bottom of the Schema Map editor's workspace.

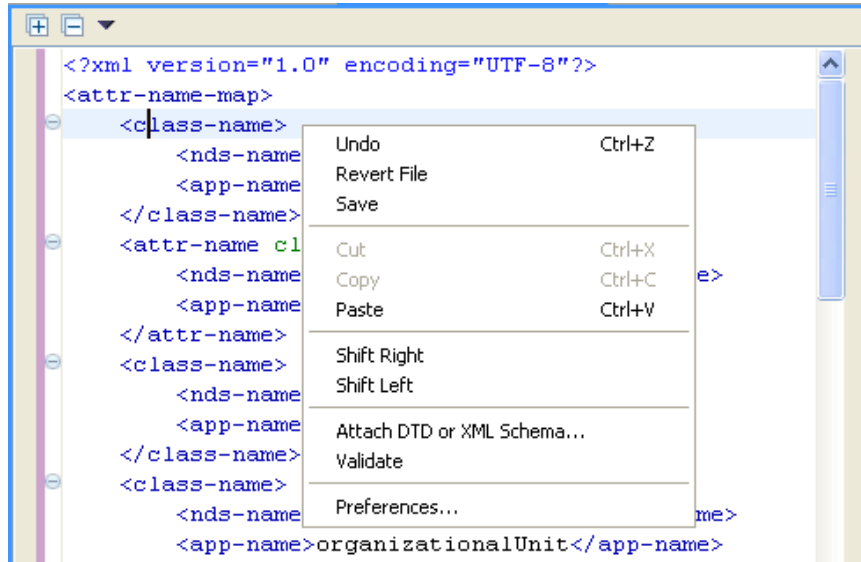


To see the entire tree view, expand each item listed.

Editing the XML Source

You can edit the XML through the XML editor. You can make changes here as well as through the GUI interface.

Figure 7-6 Editing the XML Source for the Schema Map Policy



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 of the XML Source view is based on the editor that loads.

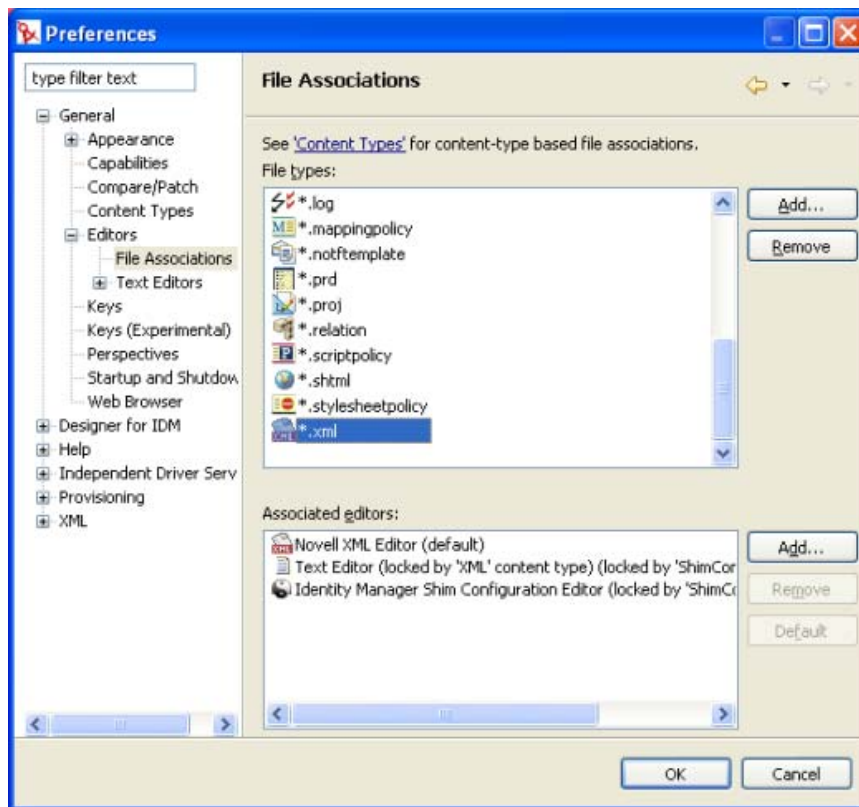
Right-click to display the list of the functions the XML editor contains.

- ◆ **Undo:** Undoes the last action.
- ◆ **Revert File** Reverts the file to the last version that was saved.
- ◆ **Saves:** Saves the file.
- ◆ **Cut:** Cuts the selected information.
- ◆ **Paste:** Pastes the information into the document.
- ◆ **Shift Right:** Indents the line to the right.
- ◆ **Shift Left:** Indents the line to the left.
- ◆ **Attach DTD or XML Schema:** Attaches a DTD or XML schema file for validation of the policy.
- ◆ **Validate:** Validates the XML code.
- ◆ **Preferences:** Sets the preferences for the XML editor.

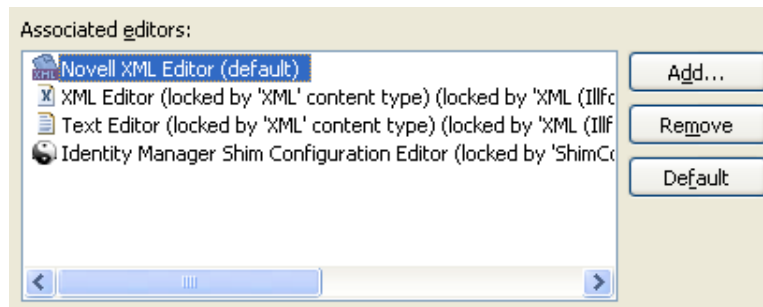
To choose a different XML editor for your source view:

- 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*) from the *Associated editors*. If the editor you want isn't in the list, you can click *Add*, then add it to the list.

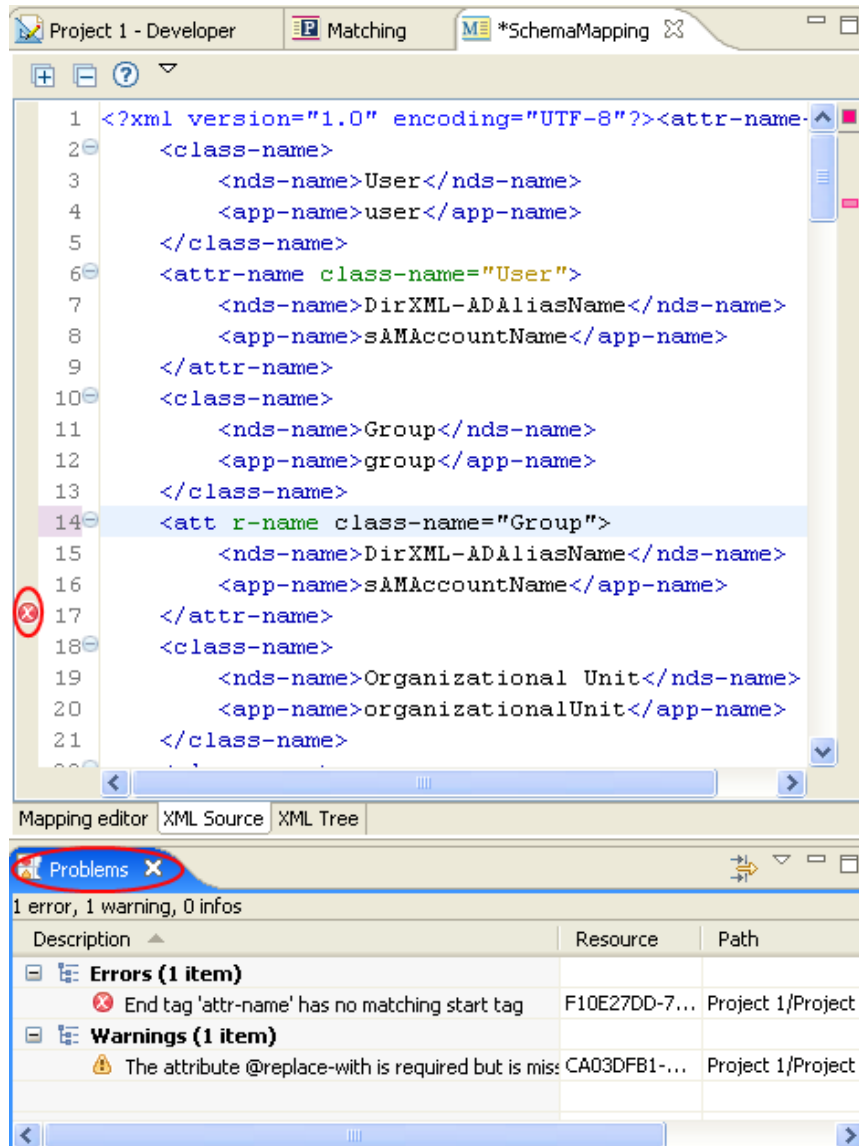


- 5 Click *OK*.
- 6 Close and reopen the Schema Map editor. The default editor should be loaded in the *XML Source* view.

Validating the XML Source

The XML editor validates the XML code. Right-click, then select *Validate*. If there are errors, a red x is displayed on the line where the error occurs. An explanation at the bottom of the window gives more information about the problem.

Figure 7-7 Validating Schema Map Policy



In this example, the end tag of `<attr-name>` has no matching start tag.

7.1.5 Additional Schema Map Policy Options

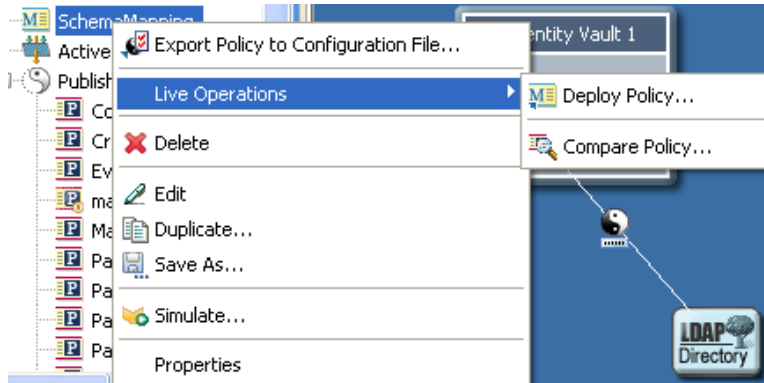
When you right-click on a Schema Map policy, there are multiple options presented in the Outline view, the Policy Flow view, and the Policy Set view.

- ◆ [“Outline View Additional Options” on page 433](#)

- ◆ “Policy Flow Additional Options” on page 434
- ◆ “Policy Set View Additional Options:” on page 435

Outline View Additional Options

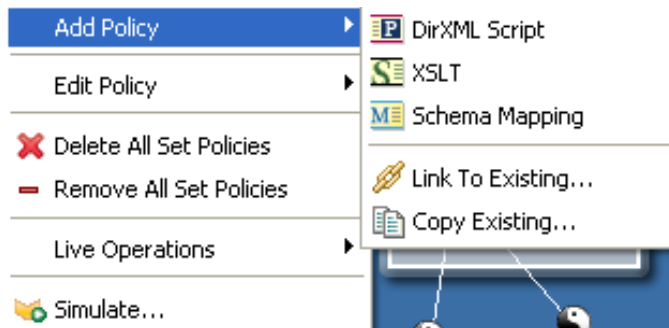
1 Right-click the Schema Map policy in the Outline view.



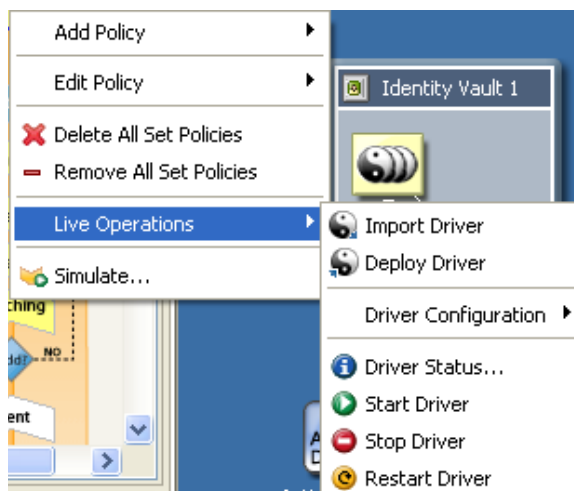
- ◆ **Export Policy to Configuration File:** Saves the Schema Map policy as a .xml file.
- ◆ **Live Operations > Deploy Policy:** Deploys the Schema Map policy into the Identity Vault.
- ◆ **Live Operations > Compare Policy:** Compares the Schema Map policy in Designer to the Schema Map policy in the Identity Vault.
- ◆ **Delete:** Deletes the Schema Map policy.
- ◆ **Edit:** Launches the Schema Map editor. For more information, see [Section 7.1.2, “Editing a Schema Mapping Policy,”](#) on page 417.
- ◆ **Duplicate:** Creates a copy of the Schema Map policy.
- ◆ **Save As:** Saves the Schema Map policy as a .xml file.
- ◆ **Simulate:** Tests the Schema Map policy. For more information, see [Section 7.1.3, “Testing Schema Mapping Policies,”](#) on page 420.
- ◆ **Properties:** Allows you to rename the Schema Map policy.

Policy Flow Additional Options

- 1 Right-click the Schema Map policy in the Policy Flow view.



- ◆ **Add Policy > DirXML Script:** Adds a new Schema Map policy using DirXML[®] Script.
- ◆ **Add Policy > XSLT:** Adds a new Schema Map policy using XSLT.
- ◆ **Add Policy > Schema Mapping:** Adds a new Schema Map policy, that contains no information.
- ◆ **Add Policy > Link to Existing:** Allows you to browse and select an existing Schema Map policy to link to the current Schema Map policy.
- ◆ **Add Policy > Copy Existing:** Allows you to browse to and select an existing Schema Map policy to copy to the current Schema Map policy.
- ◆ **Edit Policy > Schema Mapping:** Launches the Schema Map editor. For more information, see [Section 7.2.2, “Editing the Schema Mapping Policy,” on page 436](#).
- ◆ **Delete All Set Policies:** Deletes all policies in the selected policy set.
- ◆ **Remove All Set Policies:** Removes all policies from the selected policy set, but it does not delete the existing policies.

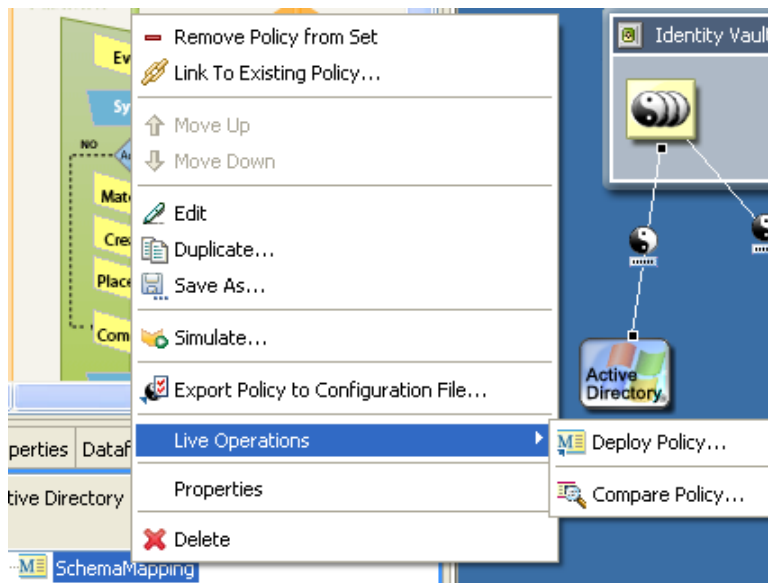


- ◆ **Live Operations > Import Driver:** Imports an existing driver from the Identity Vault.
- ◆ **Live Operations > Deploy Driver:** Deploys the existing driver into the Identity Vault.

- ♦ **Live Operations > Driver Configuration > Import Attributes:** Allows you to import attributes from the Identity Vault and compare the attributes from the Identity Vault to what is in Designer.
- ♦ **Live Operations > Driver Configuration > Deploy Attributes:** Allows you to deploy attributes from Designer into the Identity Vault and compare the attributes from Designer with the attributes in the Identity Vault.
- ♦ **Live Operations > Driver Status:** Displays the status of the driver.
- ♦ **Live Operations > Start Driver:** Starts the driver.
- ♦ **Live Operations > Stop Driver:** Stops the driver.
- ♦ **Live Operations > Restart Driver:** Restarts the driver.
- ♦ **Simulate:** Tests the Schema Map policy. For more information, see [Section 7.1.3, “Testing Schema Mapping Policies,”](#) on page 420.

Policy Set View Additional Options:

- 1 Right-click the Schema Map policy in the Policy Set view.



- ♦ **Remove Policy from Set:** Removes the Schema Map policy from the policy set, but does not delete the Schema Map policy.
- ♦ **Link to Existing Policy:** Allows you to browse to another Schema Map policy and link it into the existing policy.
- ♦ **Move Up:** Moves the Schema Map policy up in the execution order of the policy.
- ♦ **Move Down:** Moves the Schema Map policy down in the execution order of the policy.
- ♦ **Edit:** Launches the Schema Map editor. For more information, see [Section 7.2.2, “Editing the Schema Mapping Policy,”](#) on page 436.
- ♦ **Duplicate:** Creates a copy of the Schema Map policy.
- ♦ **Save As:** Saves the Schema Map policy as a .xml file.

- ♦ **Simulate:** Tests the Schema Map policy. For more information, see [Section 7.1.3, “Testing Schema Mapping Policies,”](#) on page 420.
- ♦ **Export Policy to Configuration File:** Saves the Schema Map policy as a .xml file.
- ♦ **Live Operations > Deploy the Policy:** Deploys the Schema Map policy into the Identity Vault.
- ♦ **Live Operations > Compare Policy:** Compares the Schema Map policy in Designer to the Schema Map policy in the Identity Vault.
- ♦ **Properties:** Allows you to rename the Schema Map policy.
- ♦ **Delete:** Deletes the Schema Map policy.

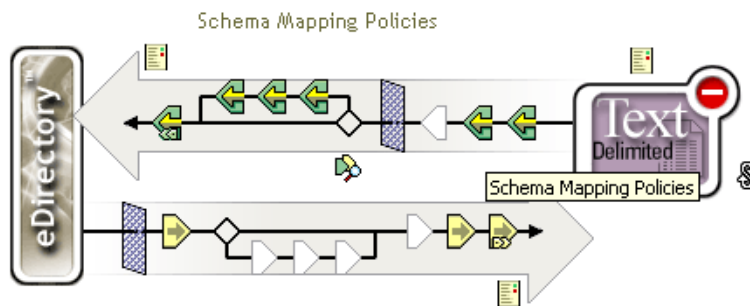
7.2 Schema Mapping Policy Tasks in iManager

This section contains instructions on performing common tasks related to Schema Mapping policies in iManager:

- ♦ [Section 7.2.1, “Accessing Schema Mapping Policies,”](#) on page 436
- ♦ [Section 7.2.2, “Editing the Schema Mapping Policy,”](#) on page 436

7.2.1 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.



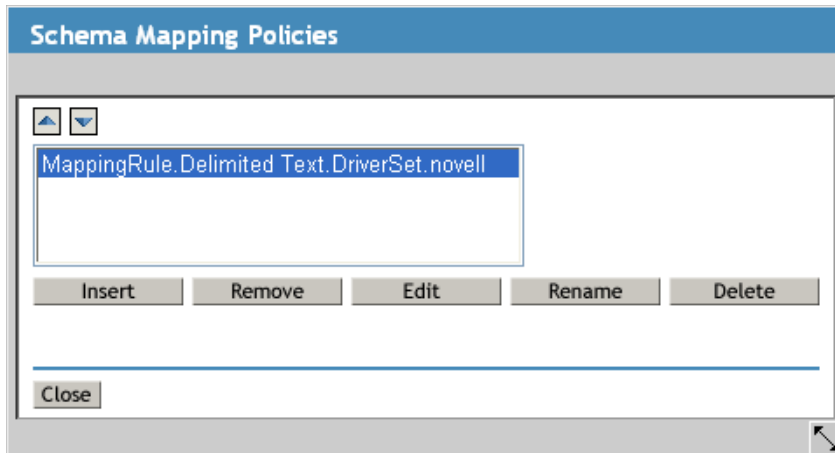
- 4 Click the Schema Mapping Policy.
- 5 Click *Edit*.

7.2.2 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.

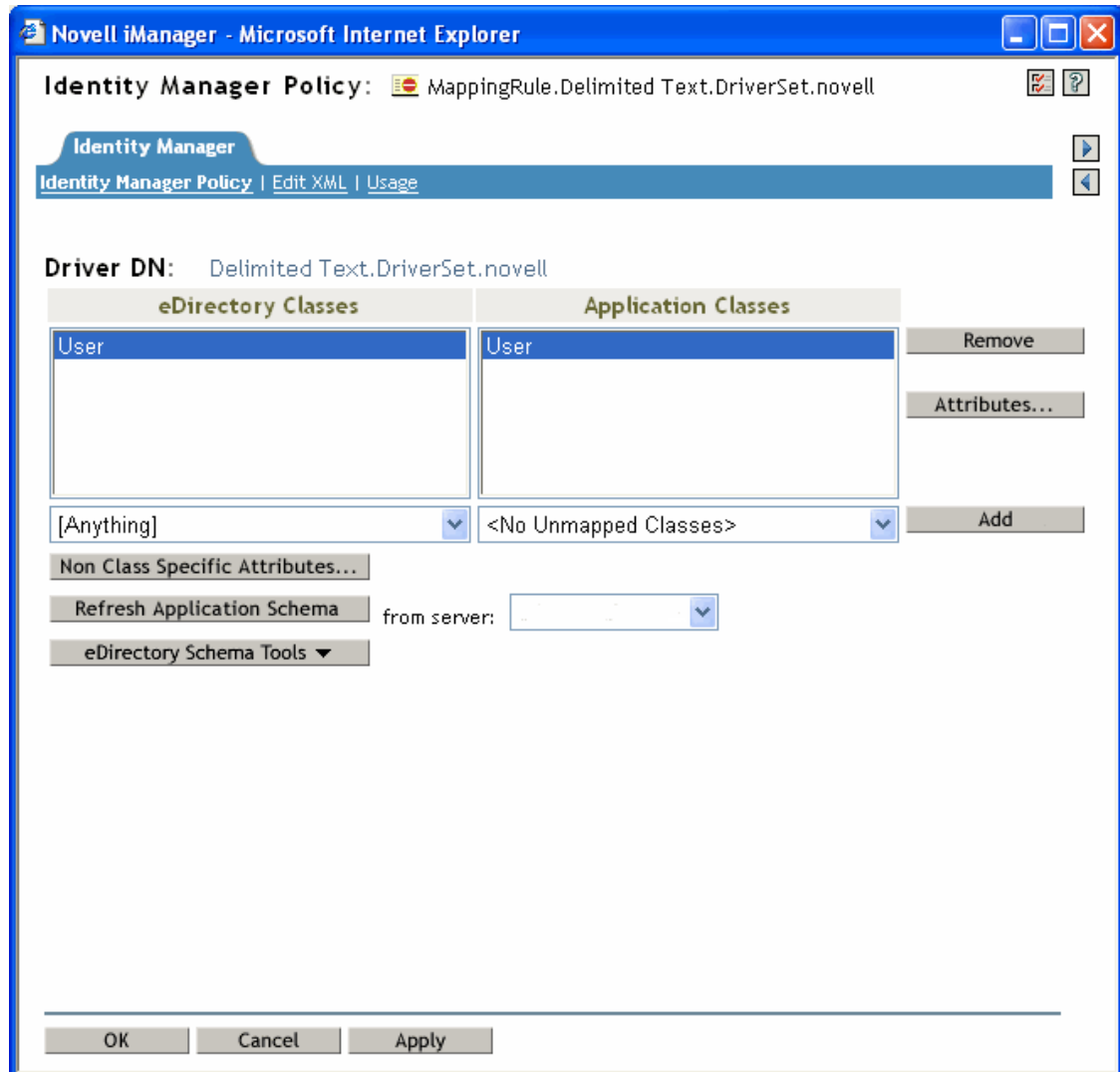


These options allow you to position the policy you are currently working with. The following table explains each of the options.

Option	Description
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 438](#)
- ◆ [“Edit XML” on page 439](#)
- ◆ [“Usage” on page 439](#)

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 <i>Apply</i> 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"> ◆ Add Attribute - Adds an existing attribute to the selected class. ◆ Create Attribute - Creates a new attribute. ◆ Create Class - Creates a new class. ◆ Delete Attribute - Deletes the selected attribute. ◆ Delete Class - Deletes the selected class. ◆ 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.

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.

Documentation Update

A

The documentation was updated on the following dates:

- ♦ Section A.1, “March 26, 2007,” on page 441
- ♦ Section A.2, “October 3, 2006,” on page 441
- ♦ Section A.3, “September 8, 2006,” on page 442
- ♦ Section A.4, “July 31, 2006,” on page 443

A.1 March 26, 2007

The following section was updated:

A.1.1 Introduction to Policies

Location	Change
Section , “Downloadable Identity Manager Policies,” on page 36	Added the procedure of how to download the policies from the Novell Support Web site.

A.2 October 3, 2006

The following sections were updated:

A.2.1 Defining Policies By Using the Policy Builder with Designer

Location	Change
Section 2.7.13, “Operation Attribute,” on page 193	Changed the definition. It is now “Expands to the value of the specified attribute from the current XDS operation. It is different from Source Attribute and Destination Attribute, because it is always accessed directly from what is available in the current XDS operation as opposed to being queried from the source or destination data stores. It does not include the removed values from a modify operation.”
Section 2.7.14, “Operation Property,” on page 194	Changed the definition. It is now “The XML attribute attached to an <operation-data> element by a policy. It is a place for policies to store and forward information for consumption by other policies. An XML attribute is a name value pair associated with an element in the XDS document.”

A.2.2 Defining Policies By Using the Policy Builder with iManager

Location	Change
Section 3.7.13, "Operation Attribute," on page 312	Changed the definition. It is now "Expands to the value of the specified attribute from the current XDS operation. It is different from Source Attribute and Destination Attribute, because it is always accessed directly from what is available in the current XDS operation as opposed to being queried from the source or destination data stores. It does not include the removed values from a modify operation."
Section 3.7.14, "Operation Property," on page 313	Changed the definition. It is now "The XML attribute attached to an <operation-data> element by a policy. It is a place for policies to store and forward information for consumption by other policies. An XML attribute is a name value pair associated with an element in the XDS document."

A.3 September 8, 2006

The following sections were updated:

A.3.1 Implementing Credential Provisioning Policies with Novell SecureLogin

Location	Change
Section 4.2.1, "Meeting Requirements for Credential Provisioning Policies with Novell SecureLogin," on page 329	Added eDirectory 8.8.1 as a supported version.

A.3.2 Configuring Credential Provisioning Policies for Novell SecureLogin

Location	Change
Section , "Example Credential Provisioning Policies," on page 347	Added the path for eDirectory 8.8.1.

A.3.3 Implementing Credential Provisioning Policies with Novell SecretStore

Location	Change
Section 4.4.1, "Meeting Requirements for Credential Provisioning Policies with Novell SecretStore," on page 352	Added eDirectory 8.8.1 as a supported version.

A.3.4 Configuring Credential Provisioning Policies for Novell SecretStore

Location	Change
Section , "Example Credential Provisioning Policies," on page 370	Added the path for eDirectory 8.8.1.

A.4 July 31, 2006

Updates were made to the following sections. The changes are explained below.

A.4.1 Introduction to Policies

The following updates were made in this section:

Location	Change
Section , "Creation Policy," on page 20	Added the last example of vetoing all users named Fred.
