

Novell Designer for Identity Manager

2.0

February 20, 2007

IDENTITY MANAGER USER
APPLICATION: DESIGN 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 © 2006 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. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.novell.com/company/legal/patents/> and one or more additional patents or pending patent applications in the U.S. and in other countries.

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

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

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

Third-Party Materials

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

Contents

About This Guide	11
1 Introduction to the User Application Design Tools	13
1.1 About the Provisioning View	13
1.2 About the Directory Abstraction Layer Editor	13
1.3 About the Provisioning Request Definition Editor	14
1.4 About the ECMA Expression Builder	14
1.5 Documenting a Project	14
2 Working with the Provisioning View	17
2.1 Setting Up a Provisioning Project	17
2.1.1 Creating a User Application Driver	17
2.1.2 About E-Mail Notification Templates	21
2.2 Accessing the Provisioning View	22
2.3 Setting Provisioning View Preferences	24
2.4 Importing Provisioning Objects	27
2.4.1 Importing from a Driver Configuration File	27
2.4.2 Importing from an Identity Vault	28
2.5 Exporting Provisioning Objects	28
2.5.1 Exporting to a Driver Configuration File	28
2.6 Validating Provisioning Objects	28
2.7 Deploying Provisioning Objects	29
2.7.1 Deploying Provisioning Objects	30
2.7.2 Testing the Deployed Changes	30
2.8 Comparing Provisioning Objects	31
2.9 Localizing Display Labels	31
2.9.1 Supported Languages	32
2.9.2 Localizing Directory Abstraction Layer Display Labels	33
2.9.3 Exporting and Importing Localized Labels	33
3 Configuring the Directory Abstraction Layer	37
3.1 About the Directory Abstraction Layer	37
3.1.1 Analyzing the User Application's Data Needs	37
3.1.2 About the Directory Abstraction Layer Editor	38
3.1.3 About Directory Abstraction Layer Editor Files	40
3.2 Working with Entities and Attributes	41
3.2.1 About Entities and Attributes	41
3.2.2 Adding Entities	42
3.2.3 Adding Attributes	47
3.2.4 Updating the Schema Elements List	49
3.3 Working with Lists	49
3.4 Working with Queries	53
3.5 Working with Relationships	56
3.6 Working with Configuration Settings	59
3.7 Directory Abstraction Layer Property Reference	60
3.7.1 Entity Properties	60

3.7.2	Attribute Properties	64
3.7.3	Queries Properties	73
3.7.4	Relationship Properties	74
4	Working with the Provisioning Request Definition Editor	77
4.1	About the Provisioning Request Definition Editor	77
4.1.1	How the Provisioning Request Definition Editor Fits into the Identity Manager Architecture	77
4.1.2	Provisioning and Workflow Example	78
4.2	Basic Steps for Creating a Provisioning Request Definition	83
4.3	Guidelines for Creating Workflows	84
4.3.1	Rules for Activities	84
4.3.2	Rules for Flow Paths	85
4.3.3	Understanding Workflow Data	86
4.4	Working with the Installed Templates	90
4.5	Debugging a Workflow	92
5	Creating a Provisioning Request Definition	95
5.1	About the Wizard and the Overview Tab	95
5.2	Using the Wizard to Create a Provisioning Request Definition	97
5.2.1	Using a Template	98
5.2.2	From Concept to Finished Product	99
5.3	Using the Overview Tab to Modify Basic Settings	100
6	Creating Forms for a Provisioning Request Definition	103
6.1	About Forms	103
6.1.1	About Form Control Data Binding	105
6.1.2	About Forms and Events	105
6.2	About the Forms Tab	107
6.2.1	About Form Selection	108
6.2.2	About Form Controls	109
6.3	Creating forms	111
6.3.1	Creating New Forms	111
6.3.2	Adding Form Controls and Actions	112
6.3.3	Defining Events	114
6.3.4	Using the Scripts Tab	118
6.4	Action Reference	120
6.5	Form Control Reference	122
6.5.1	Controls for User-Entered Comments	123
6.5.2	General Form Control Properties	124
6.5.3	CheckBoxPickList	124
6.5.4	DatePicker	125
6.5.5	DateTimePicker	126
6.5.6	DNContainer	128
6.5.7	DNDisplay	128
6.5.8	DNLookup	129
6.5.9	DNMaker	133
6.5.10	DNQuery	134
6.5.11	Global List	135
6.5.12	Html	136
6.5.13	MVCheckbox	136
6.5.14	MVEditor	137
6.5.15	PickList	143

6.5.16	Static List	145
6.5.17	Text146
6.5.18	Text Area	146
6.5.19	Title	147
6.5.20	TrueFalseRadioButtons	147
6.5.21	TrueFalseSelectBox	147
6.6	Working with Distinguished Names	147
6.6.1	Formatting DNs	148
6.6.2	Working with Object Selectors	148
6.7	Using DAL Queries in Forms	151
7	Creating the Workflow for a Provisioning Request Definition	157
7.1	About the Workflow Tab	157
7.1.1	Canvas	158
7.1.2	Palette	159
7.1.3	Views	160
7.2	Adding Activities to a Workflow	161
7.2.1	Setting the General Properties of an Activity	161
7.2.2	Defining the Data Item Mappings	162
7.2.3	Defining the Email Notification Settings	163
7.3	Adding the Flow Paths	164
7.4	Configuring Flow Paths	165
7.5	Addressing an Approval Activity	167
7.5.1	Valid Addressee Expressions	167
7.5.2	Relationship Between Addressee and Approver Type	168
7.6	Provisioning Multiple Individuals with One Workflow Instance	171
7.6.1	Basic Steps for using the Workflow	171
7.6.2	Setting up the Workflow for a Team Manager to Use	173
7.7	Working with Entity Activities	173
7.7.1	Adding or Modifying an Entity	173
7.7.2	Using an Entity Activity to Delete an Entity	174
7.7.3	Using an Entity Activity to Delete an Attribute or Value	174
7.8	Configuring Digital Signature Support	175
7.8.1	Digital Signature Workflow Properties	176
7.8.2	Creating a Signature Declaration	176
8	Workflow Activity Reference	179
8.1	Start Activity	179
8.1.1	Properties	179
8.1.2	Data Item Mapping	180
8.1.3	Email Notification	180
8.2	Approval Activity	181
8.2.1	Properties	181
8.2.2	Data Item Mapping	185
8.2.3	E-mail Notification	186
8.3	Log Activity	188
8.3.1	Properties	188
8.3.2	Data Item Mapping	188
8.3.3	E-mail Notification	189
8.4	Branch Activity	189
8.4.1	Properties	189
8.4.2	Data Item Mapping	189
8.4.3	E-mail Notification	189
8.5	Mapping Activity	189

8.5.1	Properties	190
8.5.2	Data Item Mapping	190
8.5.3	E-mail Notification	190
8.6	Merge Activity	190
8.6.1	Properties	191
8.6.2	Data item mapping	191
8.6.3	Email notification	191
8.7	Condition Activity	191
8.7.1	Properties	191
8.7.2	Data Item Mapping	192
8.7.3	Email Notification	192
8.8	Workflow Status	192
8.8.1	Properties	192
8.8.2	Data Item Mapping	192
8.8.3	E-mail Notification	193
8.9	Finish Activity	193
8.9.1	Properties	193
8.9.2	Data Item Mapping	193
8.9.3	E-mail Notification	193
8.10	Integration Activity	194
8.10.1	Properties	195
8.10.2	Data Item Mapping	195
8.10.3	E-Mail Notification	196
8.11	Entitlement Activity	196
8.11.1	Properties	197
8.11.2	Data Item Mapping	197
8.11.3	E-mail Notification	198
8.12	Entity Activity	198
8.12.1	Properties	198
8.12.2	Data Item Mapping	198
8.12.3	Email notification	200

9 Working with Integration Activities 201

9.1	About the Integration Activity	201
9.2	Adding an Integration Activity	201
9.3	Moving Data to and from the Integration Activity	203
9.4	Using the Integration Activity Editor Interface	206
9.4.1	XML Views	206
9.4.2	Action Model	210
9.4.3	WSDL Editor	217
9.4.4	Messages	217
9.4.5	Regenerating Code for the Action Model	217
9.4.6	Adding Actions to the Action Model	217
9.5	Actions	218
9.5.1	Advanced	218
9.5.2	Data Exchange	224
9.5.3	Repeat	229
9.5.4	Comment	236
9.5.5	Decision	237
9.5.6	Function	238
9.5.7	Log	239
9.5.8	Map	240

10 Working with ECMA Expressions 249

10.1	About the ECMA Expression Builder	249
------	-----------------------------------	-----

10.1.1	About ECMAScript	249
10.1.2	ECMAScript Capabilities	250
10.1.3	Using the ECMA Expression Builder	250
10.1.4	About Java Integration	256
10.1.5	About XPath Integration	257
10.1.6	Performance Considerations	258
10.2	ECMAScript Examples	259
10.2.1	General Examples	259
10.2.2	Flowdata Examples	259
10.2.3	Form Control Examples	260
10.2.4	Error Handling Examples	261
10.3	ECMAScript API	262
10.3.1	Form Action Script Methods	262
10.3.2	DOM Methods	269
10.3.3	ECMAScript Core	293
10.3.4	Global Functions	312
10.3.5	IDVault Functions	312

About This Guide

This guide describes how to use the Designer to create User Application components. It explains how to work with the provisioning view, the directory abstraction layer editor, and the provisioning request definition editor.

Audience

This guide is intended for designers responsible for creating workflow-based provisioning applications that run on Identity Manager.

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.

Additional Documentation

For documentation on other Identity Manager features, see the [Identity Manager Documentation Web site](http://www.novell.com/documentation/idm) (<http://www.novell.com/documentation/idm>).

Documentation Conventions

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

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

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a backslash. Users of platforms that require a forward slash, such as Linux or UNIX, should use forward slashes as required by your software.

Introduction to the User Application Design Tools

1

This section provides an overview of the tools available for designing the User Application. Topics include:

- ♦ [Section 1.1, “About the Provisioning View,” on page 13](#)
- ♦ [Section 1.2, “About the Directory Abstraction Layer Editor,” on page 13](#)
- ♦ [Section 1.3, “About the Provisioning Request Definition Editor,” on page 14](#)
- ♦ [Section 1.4, “About the ECMA Expression Builder,” on page 14](#)
- ♦ [Section 1.5, “Documenting a Project,” on page 14](#)

1.1 About the Provisioning View

The Provisioning view provides persistent access to Designer’s provisioning features. Use the Provisioning view to perform these actions on provisioning objects:

- ♦ Import object definitions from the Identity Vault or the local file system.
- ♦ Export object definitions to the local file system.
- ♦ Validate local object definitions.
- ♦ Deploy object definitions to the Identity Vault.
- ♦ Compare the objects on the local file system with those in the Identity Vault.
- ♦ Access the directory abstraction layer editor.
- ♦ Access the provisioning request definitions editor.

Double-clicking an item from the Provisioning view opens the editor for that item.

1.2 About the Directory Abstraction Layer Editor

The directory abstraction layer editor allows you to define directory abstraction layer definitions. Use the directory abstraction layer editor to modify the User Application’s behavior by:

- ♦ Adding new entities (Identity Vault objects).
- ♦ Defining the set of attributes for an entity.
- ♦ Specifying the contents of lists.
- ♦ Modeling relationships among entities.
- ♦ Defining automatic lookups between entities.
- ♦ Defining LDAP searches as Queries that you can run from request and approval forms.

1.3 About the Provisioning Request Definition Editor

The provisioning request definition editor allows you to create custom provisioning request definitions by using a rich set of Eclipse-based design tools. Use the provisioning request definition editor to:

- ♦ Define the basic characteristics of the provisioning request.
- ♦ Design the associated workflow.
- ♦ Define the request and approval forms.
- ♦ Configure the activities and flow paths.

1.4 About the ECMA Expression Builder

Designer incorporates an EMCAScript interpreter and expression editor, which allows you create script expressions that refer to and modify workflow data. For example, you can use scripting to:

- ♦ Create new data items needed in a workflow under the flowdata element.
- ♦ Perform basic string, date, math, relational, concatenation, and logical operations on data.
- ♦ Call standard or custom Java* classes for more sophisticated data operations.
- ♦ Use expressions for runtime control to:
 - ♦ Modify or override form field labels.
 - ♦ Initialize form field data.
 - ♦ Customize e-mail addresses and content.
 - ♦ Set entitlement grant/revoke rights and parameters.
 - ♦ Evaluate any past activity data to conditionally follow a workflow path using the Condition Activity.
 - ♦ Write different log messages that are conditionally triggered using a single Log Activity.

1.5 Documenting a Project

Designer provides a document generator that helps you quickly generate customized documentation for your Designer projects. You can define your own document style, but Designer ships with a default provisioning style. The default provisioning style includes sections for the User Application, such as the directory abstraction layer and the provisioning request definitions. The directory abstraction layer documentation includes the following sections:

- ♦ **Entities:** Including access properties, auxiliary classes, and LDAP classes.
- ♦ **Global lists:** Including key and display label.
- ♦ **Queries:** Including the query's keys, parameters, and conditions.
- ♦ **Relationships:** Including key, parent key, parent attribute, child key, and child attribute
- ♦ **Configuration:** Including default entity key, default locale, and container classes.

The documentation for a provisioning request definition includes:

- ♦ A table containing the definition's category, status, and e-mail notification.

- ♦ An image of the workflow's structure.
- ♦ A section for each activity with a table that lists the data mappings for the activity or the expression (if supported by the activity type).
- ♦ A section for each form.

Working with the Provisioning View

2

This section provides details on using the Provisioning view. Topics include:

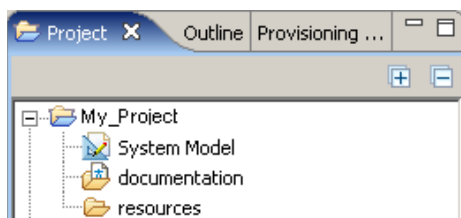
- ♦ [Section 2.1, “Setting Up a Provisioning Project,” on page 17](#)
- ♦ [Section 2.2, “Accessing the Provisioning View,” on page 22](#)
- ♦ [Section 2.3, “Setting Provisioning View Preferences,” on page 24](#)
- ♦ [Section 2.4, “Importing Provisioning Objects,” on page 27](#)
- ♦ [Section 2.5, “Exporting Provisioning Objects,” on page 28](#)
- ♦ [Section 2.6, “Validating Provisioning Objects,” on page 28](#)
- ♦ [Section 2.7, “Deploying Provisioning Objects,” on page 29](#)
- ♦ [Section 2.8, “Comparing Provisioning Objects,” on page 31](#)
- ♦ [Section 2.9, “Localizing Display Labels,” on page 31](#)

2.1 Setting Up a Provisioning Project

The Provisioning view is only available for projects that contain a User Application driver. After you set up an Identity Manager project (see “Creating a Project” in the *Designer for Identity Manager 3: Administration Guide*) and configure an Identity Vault and driver set for the project, you add and configure a User Application driver.

2.1.1 Creating a User Application Driver

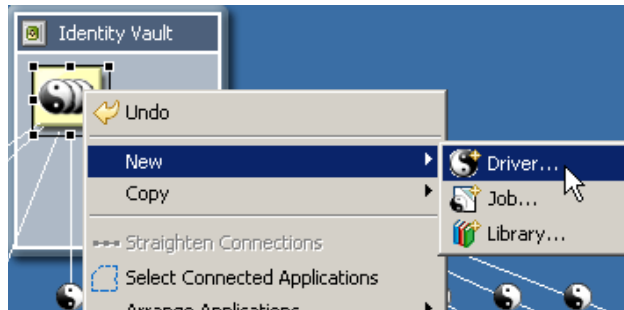
- 1 Expand the project in Project view.



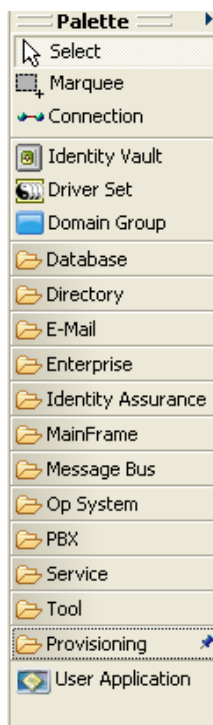
- 2 Double-click *System Model*.

3 Access the driver configuration page for a new driver using one of these methods:

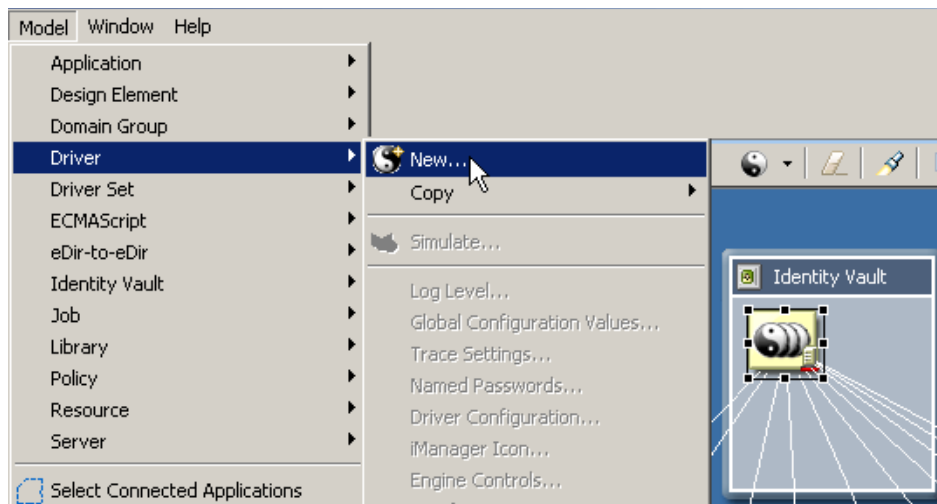
- ♦ Right-click the driver set for your project and select *New > Driver*.



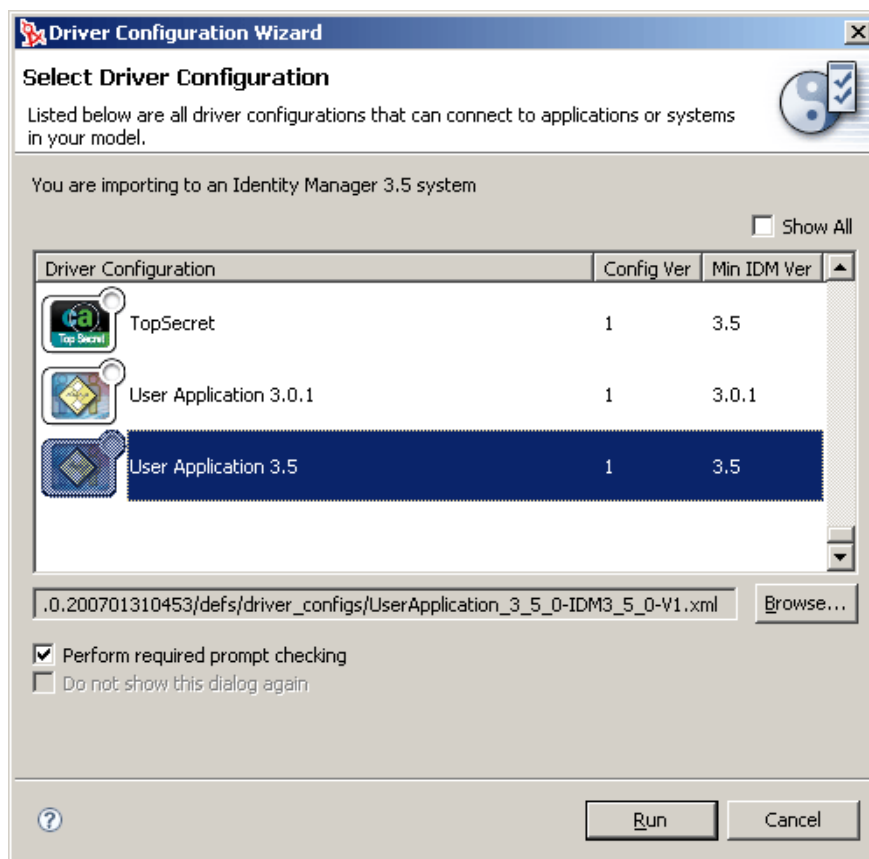
- ♦ Click *Provisioning* in the *Palette*, then drag a *User Application* icon onto the canvas.



- Click the driver set for your project and select *Model > Driver > New*.



Designer displays the Driver Configuration Wizard.

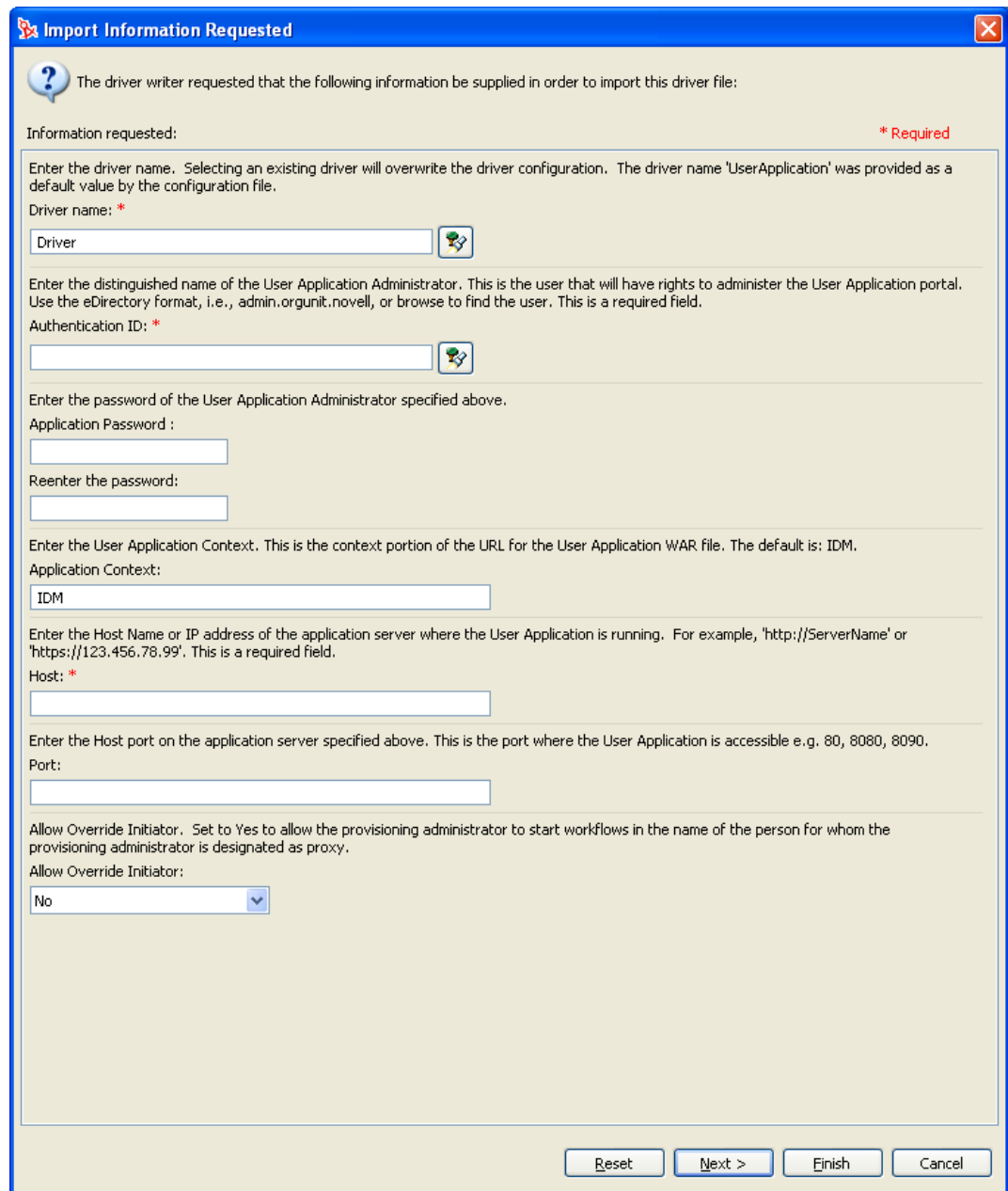


- Select one of these driver configurations.

Driver Configuration	Description
UserApplication_3_0_1.xml	Creates a Version 3.0.1 User Application driver User Application driver.
UserApplication_3_5_0.xml	Creates a Version 3.5 User Application driver.

This procedure describes how to configure both versions and indicates where fields are specific to one version or another.

Designer launches the following:




Import Information Requested

The driver writer requested that the following information be supplied in order to import this driver file:

Information requested: * Required


Enter the driver name. Selecting an existing driver will overwrite the driver configuration. The driver name 'UserApplication' was provided as a default value by the configuration file.

Driver name: *



Enter the distinguished name of the User Application Administrator. This is the user that will have rights to administer the User Application portal. Use the eDirectory format, i.e., admin.orgunit.novell, or browse to find the user. This is a required field.

Authentication ID: *



Enter the password of the User Application Administrator specified above.

Application Password :

Reenter the password:

Enter the User Application Context. This is the context portion of the URL for the User Application WAR file. The default is: IDM.

Application Context:

Enter the Host Name or IP address of the application server where the User Application is running. For example, 'http://ServerName' or 'https://123.456.78.99'. This is a required field.


Host: *

Enter the Host port on the application server specified above. This is the port where the User Application is accessible e.g. 80, 8080, 8090.

Port:

Allow Override Initiator. Set to Yes to allow the provisioning administrator to start workflows in the name of the person for whom the provisioning administrator is designated as proxy.

Allow Override Initiator:



5 Fill in the fields as follows:

Property	What to Specify
Driver Name	The name of an existing User Application driver (the driver specified during the User Application installation), or the name of a new User Application driver.
Authentication ID	The DN of the User Application Administrator.
Application password/Reenter password	The password for the User Application Administrator (above).
Application context	The name of the User Application context, for example, IDM.
Host	The host name or IP address of the application server where the Identity Manager User Application is deployed. This information is used: <ul style="list-style-type: none">♦ To trigger workflows on the application server to connect to access workflows (terminate, retract, and so on).♦ To update cached data definitions.
Port	The port for the Host (above).
Allow Override Initiator	Applies to <code>UserApplication_3_5_0.xml</code> only. This property applies to workflows that are started automatically. Typically workflows started automatically are started under the Admin identity. Selecting Yes for this property allows those workflows to be started under another user identity. For more information, see the <i>Identity Manager User Application: Administration Guide</i> .

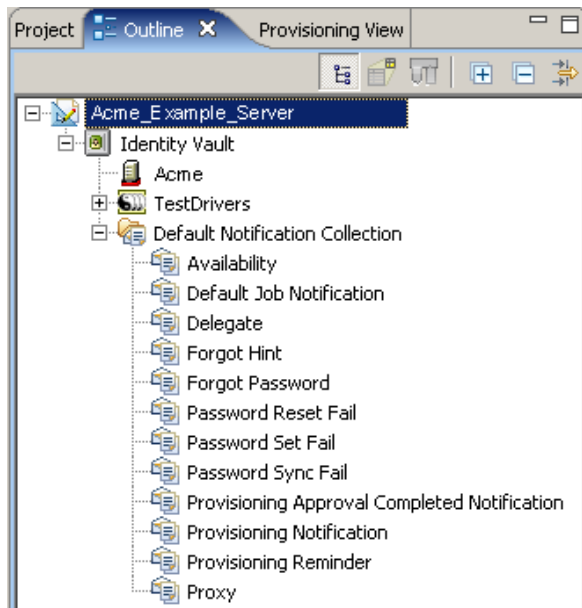
6 Click *Finish*.

2.1.2 About E-Mail Notification Templates

Identity Manager includes a standard set of e-mail notification templates (see “Working with E-Mail Templates” in the *Identity Manager 3.5 User Application: Administration Guide*). When you create a User Application driver, any e-mail notification templates that are missing from the standard set

are replaced. However, existing e-mail notification templates, which might come from an earlier version of Identity Manager, are not updated. To replace existing templates with new templates:

- 1 Expand the Outline view.



- 2 In the *Default Notification Collection*, delete the e-mail notification templates that you want to replace.
- 3 Right-click *Default Notification Collection* and select *Update Templates*.
You can also use this command at any time to update e-mail notification templates without creating a new User Application driver.
- 4 To deploy the e-mail notification templates to the Identity Vault, right-click *Default Notification Collection* and select *Live > Deploy*.

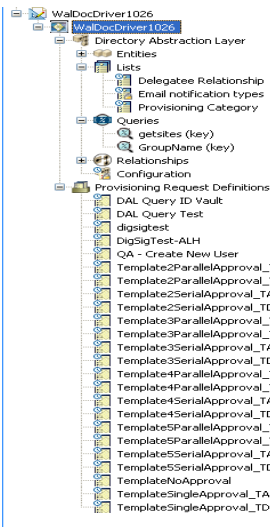
2.2 Accessing the Provisioning View

You can access the Provisioning view in the following ways:

- Select *Window > Show View > Provisioning View*.
- In the Modeler window, right-click *User Application*, then select *Show View > Provisioning View*.
- In the Outline view, right-click *User Application*, then select *Show Provisioning View*.
- Select *Provisioning View* from the FastView toolbar.




When it is open, the Provisioning view displays all of the provisioning projects located in the same workspace.

Figure 2-1 Sample Provisioning View



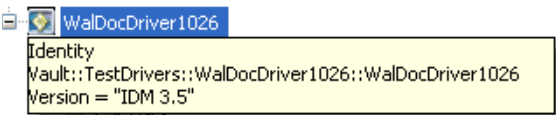
The Provisioning view displays icons to indicate the object’s status. The icons are described in the Table 2-1.

Table 2-1 Provisioning View Status Icons

Icon	Description
	Indicates that the local object has changed.
	Indicates that the local object contains a validation warning.
	Indicates that the local object contains a validation error.

Driver Information: The User Application driver icon includes a tooltip that provides the project's Identity Vault name, the DriverSet, the driver name, and the version as shown in Figure 2-2.

Figure 2-2 User Application Driver Tooltip



TIP: If you do not see the User Applications that you expect, it might be because the project is corrupt. If your project is corrupt, you must re-create it.

2.3 Setting Provisioning View Preferences

You can customize some Provisioning view behaviors by setting preferences. You access the preferences page through *Windows > Preferences > Provisioning*. The preferences include:

Table 2-2 *Provisioning View Preferences*

Preference Category	Setting	Description
General	<i>Prompt for deletion of User Application Configuration</i>	When this is selected and you delete a User Application from the Modeler, Designer asks whether you want to delete the provisioning objects on disk as part of the delete operation. If you do not delete the provisioning objects, they are left on disk, even though the User Application is deleted.
	<i>Set delete from Identity Vault as default for all "Confirm Delete" dialogs</i>	When you delete an object in the Provisioning view or the directory abstraction layer editor, you are prompted to confirm the deletion. This preference determines whether the check box labeled <i>Delete object in Identity Vault on deploy</i> in the confirmation dialog box is selected by default. Selecting this preference means that the check box is selected and the default is to delete the Identity Vault object. The local object is always deleted.
	<i>Show Provisioning View when new User Application is created or imported</i>	Select this option if you want Designer to launch the Provisioning view when you create a new User Application driver or import an existing one.

Preference Category	Setting	Description
Import/ Deploy	Import	<i>Delete local object on import when object has been deleted in Identity Vault:</i> Select this option if you want Designer to delete local objects if the corresponding Identity Vault objects were deleted. This ensures that the Identity Vault and local files are in sync. Deselect this option if you want to leave the local files alone.
	Import	<i>Prompt whether to overwrite runtime configuration on import from file:</i> Select this option if you are importing the driver from a test environment and want to deploy to a production environment. The User Application driver runtime relies on objects stored in the driver that you are not able to access in Designer. If you deploy a driver that does not contain these objects, it does not work properly. Deselect this option if you are importing the driver, modifying it, and deploying it back to the same driver set because the driver already has the runtime configuration objects.
Import/ Deploy	Deploy	<i>Allow deployment of objects with validation errors:</i> Select this option if you want to deploy objects that fail validation checks. At deployment, Designer validates the definitions being deployed following the validation rules outlined in Section 2.6, "Validating Provisioning Objects," on page 28 . Deselect this option to prevent deployment of definitions that fail validation.
		WARNING: Deploying objects that fail validation can result in errors in the User Application runtime.

Preference Category	Setting	Description
Migration	<i>Show warning about Identity Vault schema changes</i>	When you select the <i>Migrate</i> command, Designer displays a dialog box that warns you that schema changes (that are needed to support new features) must have been made before you can deploy the migrated driver. If the updates have not been made, you should cancel the migration until they are complete. If you don't want to see this warning when you select the <i>Migrate</i> command, deselect this option.
	<i>Always deploy (un-deployed) User Application Driver</i>	This preference is only relevant for User Application drivers that haven't been deployed to the directory (for example, User Application drivers that have been imported from a driver configuration file). When a User Application driver that has not been deployed is migrated, an additional dialog box is displayed that prompts you to deploy the driver. Select the <i>Always deploy (un-deployed) User Application driver</i> option if you always want Designer to deploy the User Application driver, and you don't want this dialog box displayed.
	<i>Show warning that editors will be closed</i>	When you select the <i>Migrate</i> command, Designer displays a dialog box that warns you that all editors will be closed. Select this option if you don't want this warning displayed each time you choose the <i>Migrate</i> command.
Validation Mask	<i>Validation Mask Table</i>	<p>Use this dialog to define the validation masks available to form controls such as the Text control. The validation masks are regular expressions and must follow regular expression syntax.</p> <p>Designer provides a default set of validation masks. If they do not show up as validation masks in the form controls property sheets, you can enable them by clicking <i>Restore Defaults</i>, then <i>Apply</i>.</p>

Preference Category	Setting	Description
Workflows	<i>Form Templates</i>	Use this dialog box to remove or preview existing form templates.
	<i>Diagram Preferences</i>	<p>Show Activity ID: Select this preference when you want the Workflow tab of the provisioning request definition editor to display the Activity IDs for each activity in the flow. Activity IDs are used by the ECMA expression builder and are written to the User Application's error logs.</p> <p>Show Flow Path Types: Select this preference when you want the Workflow tab of the provisioning request definition editor to display the Flow Path Types for each activity in the flow. Flow Path Types are used by the ECMA expression builder and are written to the User Application's error logs.</p>
	<i>Connection</i>	This is the amount of time (in milliseconds) for Designer to connect to the Identity Vault. If this is set too low, you might encounter an error when trying to set Trustee Rights on a provisioning request definition or when trying to access the Identity Vault via the ECMA expression builder.

2.4 Importing Provisioning Objects

The Provisioning view's import feature lets you import provisioning objects from:

- ♦ A driver configuration file
- ♦ An Identity Vault

This feature is useful when you begin a new project based on one or more definitions from an existing project, or when you want to share definitions with other developers working on the same project.

NOTE: When you change the Identity Vault or driver set's deploy context, you must save the project before performing an import. If you do not save the change, Designer continues to use the old deploy context for import operations.

2.4.1 Importing from a Driver Configuration File

To import objects from a driver configuration file:

- 1 Open the Provisioning view.
- 2 Select the root node representing the type of object you want to import.
- 3 Right-click the container and select *Import from File*. Confirm the import operation (which might overwrite existing definitions of the same name) by clicking *OK*.

- 4 Specify the name of the driver configuration file you want to import, then click *OK*.

2.4.2 Importing from an Identity Vault

- 1 Open the Provisioning view and select the container into which you want to import the definitions.

To import a specific provisioning object, select that node in the Provisioning view. To import all objects of a specific type, select the root node representing that type.

- 2 Right-click the container and select one of the following:
 - ♦ *Import Object* to import the specified object and its children.
 - ♦ *Import All* to import all of the objects of a selected container.

If prompted, provide the Identity Vault credentials and click *OK*.

- 3 Navigate to the Identity Vault container or object that you want to import and click *OK*.
- 4 Review the Import Summary page to determine how you want to proceed. To complete the import, click *Import*, or click *Cancel*. If you click *Import*, Designer performs the operation and displays a summary of the completed operation.

2.5 Exporting Provisioning Objects

The Provisioning view's export feature allows you to move project components from one project to another without re-creating the contents. It also allows you to clone a project. You can use it to export provisioning objects (and their children) to an XML-based driver configuration file. You use the resulting file as the input to the Import from File feature enabling you to easily share the contents of your provisioning project with other developers.

2.5.1 Exporting to a Driver Configuration File

- 1 Open the Provisioning view and select the object containing the definitions to export.

To export a specific provisioning object, select that node in the Provisioning view. To export all of the objects of a specific type, select the root node representing that type.

- 2 Right-click the container or object and select *Export to File*.
- 3 Provide the name and location of the file to generate, then click *OK*.

The default name for the file reflects the contents of the file. For example, if you export lists, the default name for the file is `lists.xml`. You can change the name as needed.

2.6 Validating Provisioning Objects

The Validation feature allows you to validate provisioning objects on the local file system before you deploy. The validation runs Designer's project checker and displays the results in the Project Checker view.

For directory abstraction layer objects, Designer does the following:

- ♦ Verifies that the XML is well-formed and complies with the schema that defines the elements needed for entities, attributes, lists, relationships, and so on.
- ♦ Checks every entity to ensure that references to other entities and global lists are valid.

For example, when validating an entity and its attributes, the validator checks that all references to other entities via the Edit Entity, DNLookup, and Detail Entity references exist.

- ◆ Ensures that every entity has at least one attribute defined.
- ◆ Ensures that every local and global list contains at least one item.

For Provisioning Request Definitions, Designer does the following:

- ◆ Validates that every Provisioning Request Definition has at least one request form and one approval form.
- ◆ Ensures that the Condition Activity has both an outbound true flow path and an outbound false flow path.
- ◆ Ensures that the Entitlement Activity Data Item Mapping of DirXML-Entitlement-DN is valid.
- ◆ Ensures that the Final Timeout Action property (for User Activities) has a matching flow path link leading from the activity. For example, if Final Timeout Action=denied, there must be a denied link.
- ◆ For Branch and Merge activities, ensures that a workflow has an equal number of Branch and Merge activities. It also ensures that all paths descending from a Branch activity merge into one Merge activity, that all merge activities have a branch activity, and that all Merge activities have a branch-activity-id attribute.
- ◆ Ensures that static list keys contain the correct data for the decimal data type.

To validate objects from the Provisioning view, right-click a node and click *Validate*.

To validate objects from the directory abstraction layer editor, click *Validate Abstraction Layer* from the editor's toolbar, or select *DAL > Validate* from Designer's menu.

To validate objects from the provisioning request definition editor, select *PRD > Validate* from Designer's menu.

NOTE: Validation does not check the Identity Vault for the existence of any object.

2.7 Deploying Provisioning Objects

The Provisioning view's Deploy feature deploys your provisioning objects to the specified User Application driver. You must deploy any changes you've made to the provisioning objects in the design environment before you see them reflected in the Identity Manager User Application. The Provisioning view allows you to deploy a container and all its children (for example, all entities or all lists), or to deploy just a single provisioning object (such as a single list element). When you select an item to deploy, Designer compares it to the same item in the Identity Vault. If they are equal, Designer prevents you from deploying. When there are differences, Designer displays them and allows you to proceed or to cancel the deployment.

If you deploy a Version 3.5 User Application driver and the Identity Vault does not contain the necessary 3.5 schema changes, the provisioning objects are not deployed and Designer displays an error message in the Deploy Results dialog box. This is to prevent you from deploying a 3.5 driver to a 3.0.1 Identity Vault.

NOTE: When you change the Identity Vault or driver set's deploy context, you must save the project before performing a deploy. If you do not save the change, Designer continues to use the old deploy context for deploy operations.

2.7.1 Deploying Provisioning Objects

1 Save any changes.

If the objects contain unsaved changes, Designer displays the unsaved definitions and prompts you to save them. If you do not, Designer still deploys the objects but does not deploy the unsaved changes. Choosing not to save the changes does not cancel the deployment.

2 Open the Provisioning view, right-click the object to deploy, then and select *Deploy* or *Deploy All*.

To deploy a specific provisioning object, select that node in the Provisioning view. To deploy all of the objects of a specific type, select the root node representing that type.

Designer prompts you for Identity Vault credentials (if necessary), validates the objects, and writes any messages to the project checker view.

When you deploy a driver and it contains provisioning objects that fail validation, Designer deploys the driver but not the invalid objects (regardless of the deployment preferences). The errors are displayed in the deployment result dialog box. When you deploy a provisioning object that contains validation errors, Designer performs the deployment based on the defined preferences and writes the errors to the Project Checker view.

Some tips for deploying Provisioning Request Definitions:

- ♦ If errors associated with activities are detected during deployment of a provisioning request definition, Designer identifies the activity in which the error occurred by activity ID. However, in the user interface, Designer by default displays activities by activity Name. To make it easier to identify the activity in an error message, you should turn on the display of activity IDs before you deploy the provisioning request definition. To turn on the display of activity IDs, right-click the Workflow canvas and select *Show Activity IDs*.
- ♦ A common error occurs when you fail to replace a placeholder expression in an entitlement provisioning activity (see [Section 8.11, “Entitlement Activity,” on page 196](#) for information about entitlement provisioning activities). If this is the case, correct the error, then try deploying the provisioning request definition again.
- ♦ Designer cannot evaluate expressions at design time, so you might receive a warning if you’ve used an expression for an entitlement that must be resolved at runtime. This is not a fatal error and the deployment will succeed.
- ♦ Make sure that the *Status* is *Active* (in the *Overview* tab).
- ♦ If the provisioning request definition with the same CN already exists in the Identity Vault, the Deployment Summary displays the differences, and allows you to examine the differences before you decide to proceed.

2.7.2 Testing the Deployed Changes

You can access the User Application from within Designer to view or test what you deploy. To access the User Application from Designer:

- 1 Select *Tools>Access User Application*.
- 2 Choose the project and User Application driver container associated with the User Application you want to view, then click *OK*.

Designer uses the driver configuration information, that you defined for the project, to make the connection. Designer uses the browser settings specified in *Windows>Preferences>General>Web Browser*

2.8 Comparing Provisioning Objects

The Provisioning view's Compare feature allows you to see the differences between the provisioning objects in the local file system and those that are running in the deployed User Application driver. When Designer encounters a difference, it allows you to specify what action you want to take on that difference. You can ignore or reconcile it.

NOTE: When you change the Identity Vault or DriverSet's deploy context, you must save the project before performing a compare. If you do not save the change, Designer continues to use the old deploy context for compare operations.

To compare provisioning objects:

- 1 Right-click a container or object in the Provisioning view, then select *Compare*.
- 2 If prompted, provide Identity Vault credentials, then click *OK*.
Designer displays the results of the comparison. By default, only the differences are displayed, but you can show the full comparison by deselecting *Only show differences*.
- 3 If there are differences, select one of the following actions:

Reconcile Status	Description
Do not reconcile	Do not change any definitions.
Update Designer	Import the definitions from the Identity Vault.
Update eDirectory	Deploy the definition from Designer to the Identity Vault.
Reconciled by parent	For informational purposes. Specifies whether one of the parent objects is already being reconciled. It is always disabled and is only set if the parent object is already being reconciled to Designer or the Identity Vault.

2.9 Localizing Display Labels

Designer provides an easy way to localize the display labels defined in provisioning objects. You can provide localized string values whenever you see the button in [Figure 2-3](#).

Figure 2-3 *Localize Strings Button*



When you click this button, Designer displays the Localization dialog box shown in [Figure 2-4](#).

Figure 2-4 Localization Dialog Box

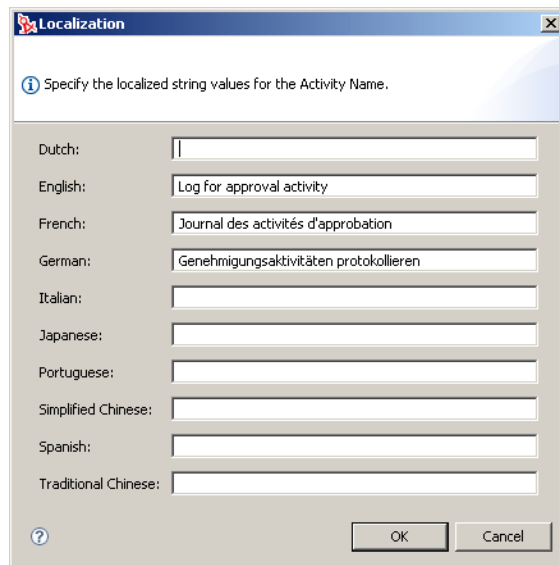


Table 2-3 Localizable Objects

Designer tool	Description
Directory Abstraction Layer Editor	<ul style="list-style-type: none">♦ Entity and attribute display labels♦ Relationship names♦ Global and local list items♦ Query display labels and parameter display labels.
Provisioning Request Definition Editor	<ul style="list-style-type: none">♦ Activity properties that are displayed to the user.♦ Form properties that are displayed to the user.

You *must* provide a display label for the User Application Driver's default language, or you will encounter the following runtime error: *The resource resolver com.novell.soa.common.i18n.LocalizedMapResolver did not return a resource for the default locale of <locale>. It is required that a resource exist for the default local.*

The locale configuration is stored in the driver's <default-locale> element in the AppConfig.AppDefs.locale-configuration XMLData attribute.

2.9.1 Supported Languages

You can localize the display labels in any language displayed in the localization dialog box.

2.9.2 Localizing Directory Abstraction Layer Display Labels

The directory abstraction layer editor provides multiple ways to localize abstraction layer definitions. You can access the localization dialog boxes in these ways:

Table 2-4 *Ways to Access the Localization Dialog Boxes*

To define the localization text for	Action
Every localizable item in the directory abstraction layer	Select <i>DAL > Set Global Localization</i> . or Click <i>Set Global Localization</i> (from the editor's toolbar), then select the <i>Target Language</i> before entering the localized text in the <i>Target</i> field.
A specific entity, relationship or list	From the tree view, right-click the object to localize, select <i>Localize</i> , then select the <i>Target Language</i> before entering the localized text in the <i>Target</i> field.
A single display label	Select a specific entity or attribute, then click <i>Localize Display Label</i> (beside the <i>Display Label</i> field in the Property pane).

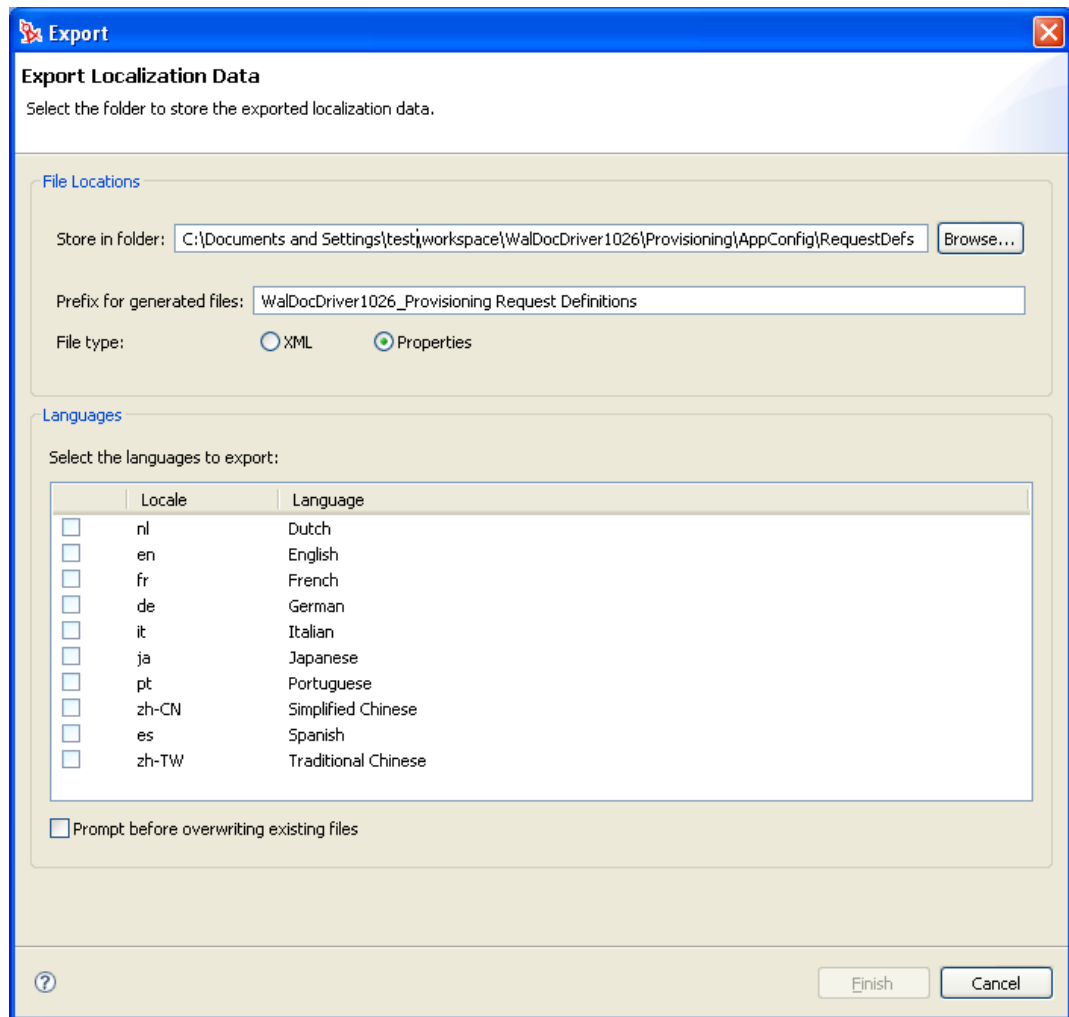
2.9.3 Exporting and Importing Localized Labels

Designer provides a wizard that lets you export all of the display labels in your User Application project to an XML or properties file that you can localize and import back. You can export an entire driver, all directory abstraction layer or provisioning request definitions, or a single object at a time.

To export display labels:

- 1 To launch the Export Localization Data Wizard, right-click on a container node or an object in the Provisioning view.

2 Select *Localize* > *Export Localization Data*.



The dialog box is titled "Export" and "Export Localization Data". It contains two main sections: "File Locations" and "Languages".

File Locations:

- Store in folder:** A text field containing "C:\Documents and Settings\test\workspace\WalDocDriver1026\Provisioning\AppConfig\RequestDefs" and a "Browse..." button.
- Prefix for generated files:** A text field containing "WalDocDriver1026_Provisioning Request Definitions".
- File type:** Two radio buttons: "XML" (unselected) and "Properties" (selected).

Languages:

Select the languages to export:

	Locale	Language
<input type="checkbox"/>	nl	Dutch
<input type="checkbox"/>	en	English
<input type="checkbox"/>	fr	French
<input type="checkbox"/>	de	German
<input type="checkbox"/>	it	Italian
<input type="checkbox"/>	ja	Japanese
<input type="checkbox"/>	pt	Portuguese
<input type="checkbox"/>	zh-CN	Simplified Chinese
<input type="checkbox"/>	es	Spanish
<input type="checkbox"/>	zh-TW	Traditional Chinese

☐ Prompt before overwriting existing files

Buttons: ? (help), Finish, Cancel

3 Fill in the fields as follows:

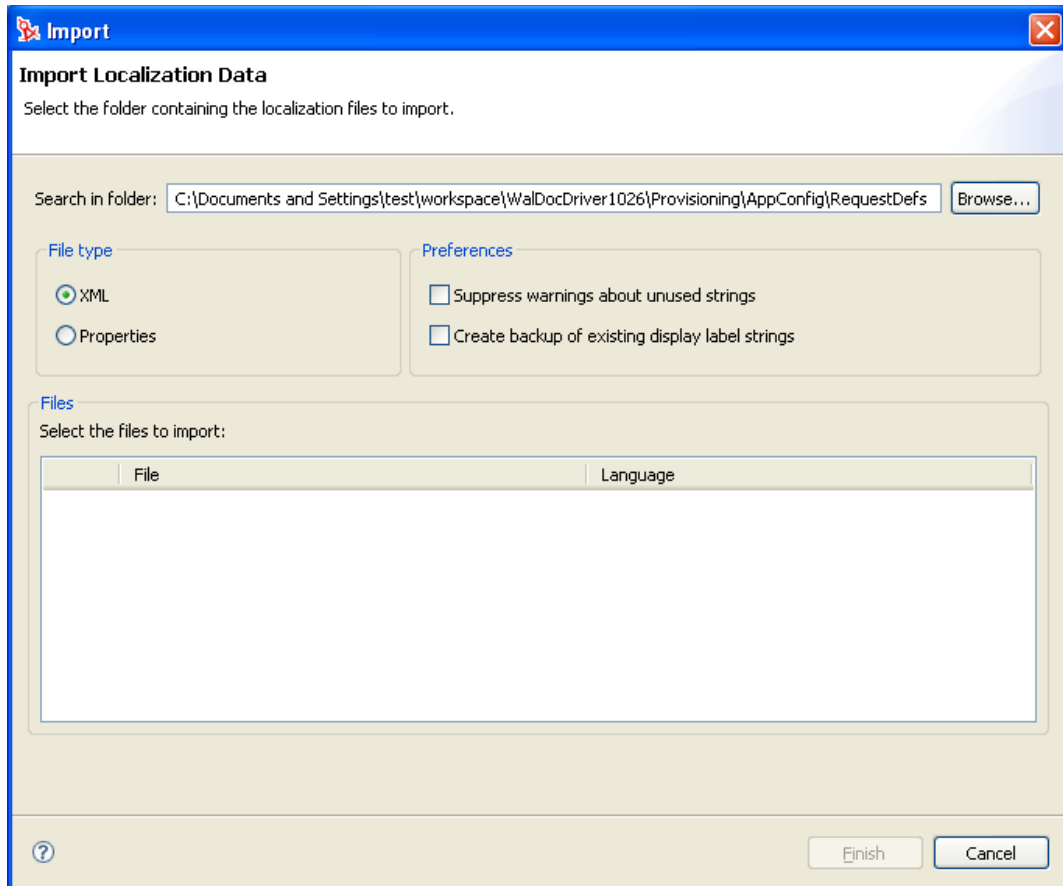
Field	Description
Store in folder	Specify the name of a local folder where the exported files should be written.
Prefix for generated files	Specify a prefix for the generated files. Determine a naming strategy so you are able to identify the files for projects.
File Type	Select <i>XML</i> or <i>Properties</i> depending on the encoding or format you prefer. XML files are UTF-8 encoded. Properties use Unicode*.

Field	Description
Select the languages to export	Select the languages you'll want localizations for. A file containing the display label key is generated for that language. The localizations need to be added to this file in the proper format so you can import them to the proper User Application driver objects.
Prompt before overwriting existing files	If this option is selected, Designer prompts you before it overwrites any existing files of the same name in the target directory.

- 4 Click *Finish*. Designer displays a message describing the result of the export operation.

To import localized files:

- 1 To launch the Import Localization Data wizard, right-click on a container node or an object in the Provisioning view.
- 2 Select *Localize > Import Localization Data*.



- 3 Fill in the fields as follows: .

Field	Description
Search in folder	Specify the folder location where the files to import are located.
File Type	<p>Select <i>XML</i> if the file you want to import is in XML format.</p> <p>Select <i>Properties</i> if the file you want to import is in the properties format.</p>
Preferences	<p>Select <i>Suppress warnings about unused strings</i> if you want the wizard to suppress warning messages.</p> <p>Select <i>Create backup of existing display label strings</i> if you want the wizard to create a backup of the existing strings before the import. Useful in case you need to revert.</p>
Files	<p>Select the files to import. This table is populated with the files from the folder location and File Type specified above. If it is blank, no files of the specified type are located in the target folder. The wizard attempts to determine the language by looking at the filename. If the name cannot be determined, it defaults to English.</p> <p>You can change the Language column if the wizard assumes the wrong language. The wizard will change the file name to reflect the language you specify and import the display labels to the corresponding language.</p>

- 4 Click *Finish* to complete the import. Designer displays a status dialog box that describes the results including any errors reading the files and any warnings about display label keys that are unused because no match was found.

Configuring the Directory Abstraction Layer

3

This section provides details on configuring the directory abstraction layer. Topics include:

- ♦ [Section 3.1, “About the Directory Abstraction Layer,” on page 37](#)
- ♦ [Section 3.2, “Working with Entities and Attributes,” on page 41](#)
- ♦ [Section 3.3, “Working with Lists,” on page 49](#)
- ♦ [Section 3.4, “Working with Queries,” on page 53](#)
- ♦ [Section 3.5, “Working with Relationships,” on page 56](#)
- ♦ [Section 3.6, “Working with Configuration Settings,” on page 59](#)
- ♦ [Section 3.7, “Directory Abstraction Layer Property Reference,” on page 60](#)

3.1 About the Directory Abstraction Layer

The directory abstraction layer is a set of XML-based files that define a logical view of an Identity Vault for the User Application. The User Application uses the directory abstraction layer definitions to determine:

- ♦ The Identity Vault objects and attributes that the User Application can display or modify.
- ♦ How the User Application displays Identity Vault data.
- ♦ The relationships the User Application can display.
- ♦ The provisioning request categories, e-mail notification types, and delegate relationships the User Application can display.

The User Application ships with a default set of entities, relationships, and lists that it needs to function, but you can add new or modify existing directory abstraction layer objects to customize the User Application for your own business needs. You use the directory abstraction layer editor (described in [Section 3.1.2, “About the Directory Abstraction Layer Editor,” on page 38](#)) to define the contents of the directory abstraction layer.

3.1.1 Analyzing the User Application’s Data Needs

Before you make changes to the directory abstraction layer objects, analyze how you want to display your Identity Vault data in the User Application. Consider:

- ♦ What parts of the Identity Vault you want to make available to the User Application.

For example, what objects do you want your users to be allowed to search and display? Check this list against the base set of abstraction layer definitions to determine if you need to add any new objects.

- ♦ What is the structure of your Identity Vault schema? Have you added custom extensions and auxiliary classes?

- ♦ What is the structure of your data?
 - ♦ What is required and what is optional?
 - ♦ What validation rules are in place?
 - ♦ What are the relationships between objects (DN references)?
 - ♦ How are the attributes defined? (For example, an attribute that represents a phone number might be multi-valued for home, office, and cell phone numbers)
- ♦ Who sees the data? Is the User Application available as a public or private site?

Use the information about your data needs to map your Identity Vault objects to abstraction layer entities.

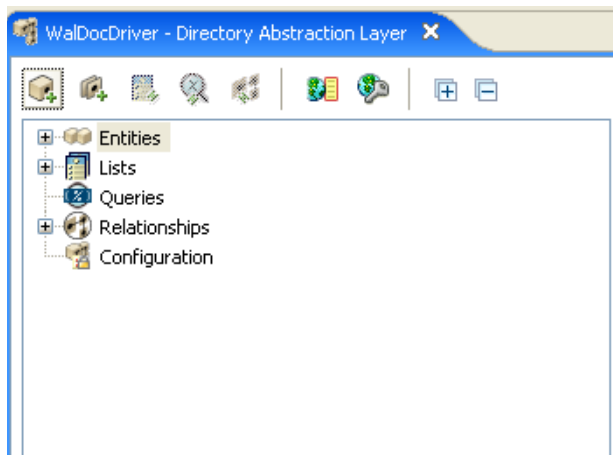
3.1.2 About the Directory Abstraction Layer Editor

The directory abstraction layer editor is a graphical tool for defining the directory abstraction layer files. When you add a User Application driver to an Identity Manager project and run the configuration wizard, Designer creates an initial set of directory abstraction layer files. If you do not run the configuration wizard, the initial files are not created. These base files are displayed when you start the directory abstraction layer editor.

To start the directory abstraction layer editor:

- 1 Open the *Provisioning* view and double-click the *Directory Abstraction Layer* node.

Designer displays the directory abstraction layer tree containing nodes for *Entities*, *Lists*, *Queries*, *Relationships*, and *Configuration*.











Node	Description
<i>Entities</i>	<p>Entities represent the Identity Vault objects available to the User Application. There are two types of entities:</p> <ul style="list-style-type: none"> ♦ Entities mapped from the schema: Entities that represent Identity Vault objects directly exposed to users via the User Application. Users can typically create, search, and modify the attributes of these entities. ♦ Entities representing LDAP relationships: Called DN lookups, these entities represent indexed searches and are used to support particular types of attributes in the User Application. DN lookup entities provide information about relationships between LDAP objects. DN lookup entities are: <ul style="list-style-type: none"> ♦ Used by the Org Chart portlet to determine relationships. ♦ Used in the Search List, Create, and Detail portlets to provide selection lists and DN contexts. ♦ Available to the workflow request and approval flow forms you define using the provisioning request definition editor.
<i>Lists</i>	<p>Defines the contents of global lists. Global lists are:</p> <ul style="list-style-type: none"> ♦ Associated with an attribute. The User Application displays the attribute values as a drop-down list in the User Application. ♦ Used to display Resource Request categories.
<i>Queries</i>	Lets you define LDAP search criteria that can be run from a workflow form.
<i>Relationships</i>	Lets you map hierarchical relationships among schema-based entities. Used by the Organization Chart action of the <i>Identity Self-Service</i> tab of the User Application and in iManager when defining provisioning teams.
<i>Configuration</i>	General configuration parameters.

- 2 Use the left pane to navigate the directory abstraction layer nodes. When you select an item in the left pane, the right pane displays the properties for the selection.
- 3 Use the right pane to define the properties for the selection. For more information about the properties, see [Section 3.7, “Directory Abstraction Layer Property Reference,”](#) on page 60.

The following table describes the directory abstraction layer toolbar:

Table 3-1 *Directory Abstraction Layer Toolbar*

Toolbar Button	Description
	Launches the Add Entity Wizard.
	Launches the Add Attribute Wizard.
	Launches the New List Wizard.
	Launches the New Query Wizard
	Launches the New Relationship Wizard.
	Launches the Set Global Access Modifiers dialog box.
	Launches the Set Global Localization dialog box.
	Expands and collapses the directory abstraction layer tree.

3.1.3 About Directory Abstraction Layer Editor Files

The directory abstraction layer files you work with are stored in the Designer project's `Provisioning\AppConfig\DirectoryModel` directory. The filenames are derived from the object key.

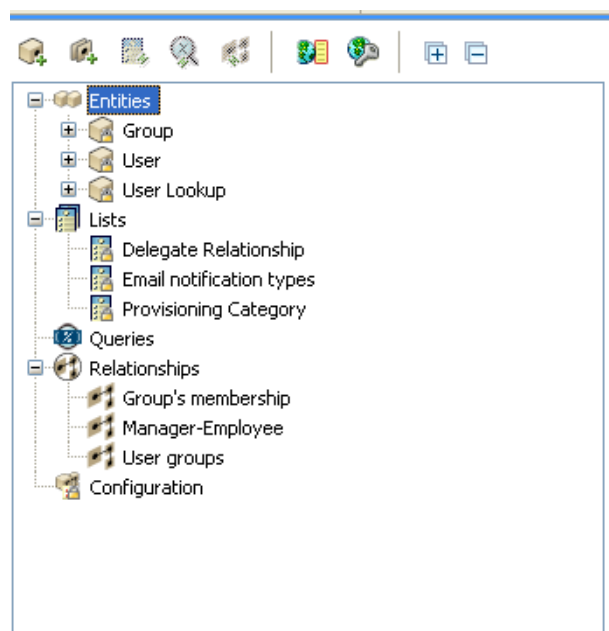
Table 3-2 *Local Directory Abstraction Layer Directories*

Directory name	Description
ChoiceDefs	Contains the files that define global lists. Files have the <code>choice</code> extension.
EntityDefs	Contains the files that define the entities and attributes. Files have the <code>entity</code> extension.
QueryDefs	Contains the files that define queries. Files have the <code>query</code> extension.
RelationshipDefs	Contains the files that define the relationships available to the Org Chart portlet and iManager provisioning teams configuration. These files have the <code>relation</code> extension.

Designer creates the base set of directory abstraction layer files for each provisioning project. An identical set is deployed to the User Application driver when the User Application is installed.

To customize the Identity Manager User Application, you change the directory abstraction layer objects and deploy the changes to the User Application driver. Some entities, attributes, lists, and relationships are required for the User Application to function properly. The editor displays a lock next to the definitions that you should not delete. From the list below, you can see that you should not delete the *Group*, *User* or *User Lookup* entities.

Figure 3-1 DAL User Application Default Entities, Lists, and Relationships



If you define multiple User Application drivers in a single project, Designer creates multiple AppConfig folders and names them AppConfig, AppConfig1, AppConfig2, and so on.

3.2 Working with Entities and Attributes

You can customize your User Application by adding objects and their attributes based on the content of your own Identity Vault. You do this by adding new entities and attributes to the directory abstraction layer and deploying them to the User Application driver.

To modify the entity files installed by default, see [Section 3.2.2, “Adding Entities,” on page 42](#) and [Section 3.2.3, “Adding Attributes,” on page 47](#). To modify the entity files of an already deployed project or a set of files defined by another developer, you must first import the files to your design environment. For information on importing files, see [Section 2.4, “Importing Provisioning Objects,” on page 27](#).

3.2.1 About Entities and Attributes

Any Identity Vault object that you want users to search, display, or edit in the Identity Manager User Application must be defined as an *entity* in the directory abstraction layer. For example, to use the

inetOrgPerson Identity Vault object in the User Application, you must create an entity definition for it. There are two logical kinds of entities (but you create them the same way):

- ♦ **Entities that are mapped from schema:** These entities represent objects that exist in the Identity Vault that are directly exposed to users in the User Application. When defining this type of entity, expose all of the attributes that you want your users to work with. Examples of this entity type include User and Group. You can create more than one entity definition for the same object to expose different sets of attributes to different kinds of users. For more information, see [“Creating Multiple Entity Definitions for a Single Object” on page 42](#).
- ♦ **Entities that represent LDAP relationships** . This type of entity is known as a *DNLookup* and it is used by the User Application to:
 - ♦ Populate a list with the results of a DN search among related entities
 - ♦ Maintain referential integrity across DN referenced attributes during updates and deletes

Entities that support DNLookups are used by the Org Chart portlet to determine relationships and are also used by the Search, Create, and Detail portlets to provide pop-up selection lists and DN contexts. The User Lookup entity is an example of this type of entity. For more information, see [“Attributes and DNLookup Properties” on page 68](#).

Creating Multiple Entity Definitions for a Single Object

You can create more than one entity definition that represents the same Identity Vault object but provides a different view of the data. Within the entity definitions, you can define different attributes for each entity definition, or you can define the same attributes but specify different access properties that control how the attributes are searched, viewed, edited, or hidden

NOTE: You can optionally define a filter to hide certain entities from the result set.

You can then use these different entity definitions in different parts of the user interface. For example, suppose that you wanted to create a directory of employees; one for a public site and one for an internal site. On the public site you wanted to supply first and last names and a phone number, but on the internal site, you wanted to list additional information like title, managers, and so on. Here’s how you can accomplish this:

- 1 Create two entity definitions (with different keys).

Both entity definitions expose the same Identity Vault object, but one entity definition key is public-staff-information, and the other is internal-staff-information.

- 2 Within each entity definition, define a different set of attributes: one for public-staff-information, the other for internal-staff-information.
- 3 Use the Portal Administration tab of the Identity Manager User Application to create a portlet instance for the public page, and another one for the internal page.

For more information about creating portlet instances, see [“Create Portlet Reference” in the *Identity Manager 3.5 User Application: Administration Guide*](#).

3.2.2 Adding Entities

You add entities through the Add Entity Wizard (described in the next procedure) or by clicking *Add Entity* (from the toolbar).

NOTE: When using the *Add Entity* button, you are prompted to select the object class of the entity to create, and the editor automatically adds the required attributes to the entity. Use the Add Attribute dialog box to complete the entity definition.

To add an entity using the Add Entity Wizard:

- 1 Launch the Add Entity Wizard in one of these ways:

From Designer's menus:

- ♦ Select *File > New > Provisioning*. Choose *Directory Abstraction Layer Entity*, then click *Next*.

From the Provisioning view:

- ♦ Right-click the *Entities* node, then choose *New*.

From the directory abstraction layer editor:

- ♦ Select *DAL > New > Entity*, or
- ♦ Right-click the *Entities* node, then choose *New Entity-Attributes Wizard*.

The New Entity dialog box displays.

NOTE: If launched from the *File* menu, the dialog box contains the additional fields shown below.

New Entity

Specify project and application for the new entity as well as the display name and key for the new entity.

Identity Manager Project: ProjectOne

Provisioning Application: User Application

Entity Key:

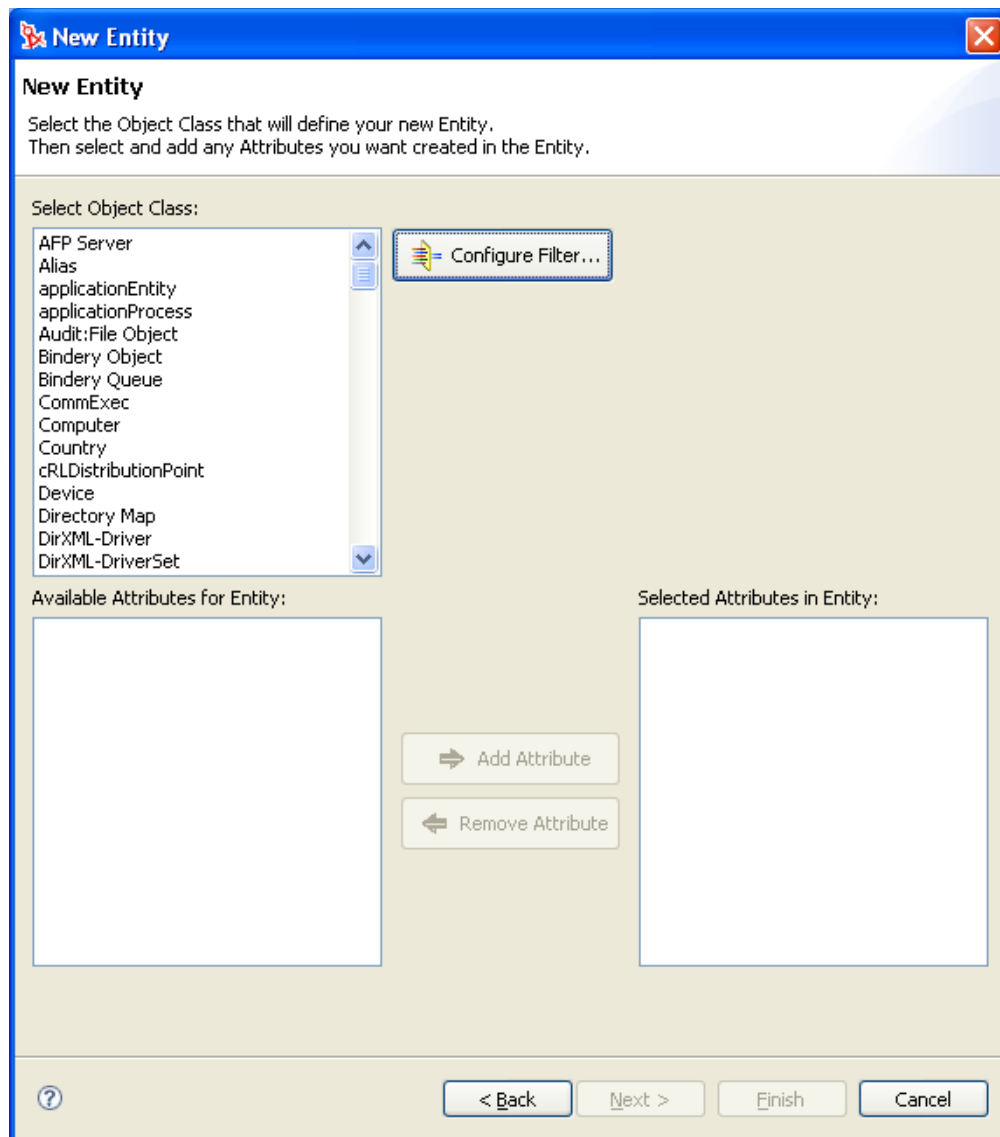
Display Label:

< Back Next > Finish Cancel

2 Fill in the fields as follows:

Field	Description
<i>Identity Manager Project and Provisioning Application</i>	The Identity Manager project and the provisioning application where you want to add the entity and attributes. NOTE: These fields display when you launch the wizard from the <i>File</i> menu.
<i>Entity Key</i>	A unique identifier for the entity.
<i>Display Label</i>	The string displayed when the entity is displayed by the User Application. You can localize this label. For more information, see Section 2.9, “Localizing Display Labels,” on page 31 .

- 3 Click *Next*. The New Entity dialog box displays:



- 4 Choose the entity's object class and add the attributes you want by double-clicking them in the *Available Attributes for Entity* list. Mandatory attributes are added when you select an *Object Class*, and you cannot remove them from the *Selected Attributes in Entity* list.

TIP: If the entity's object class is not shown in the *Select Object Class* list, you should update Designer's local schema file by following the steps described in ["Updating the Schema Elements List"](#) on page 49.

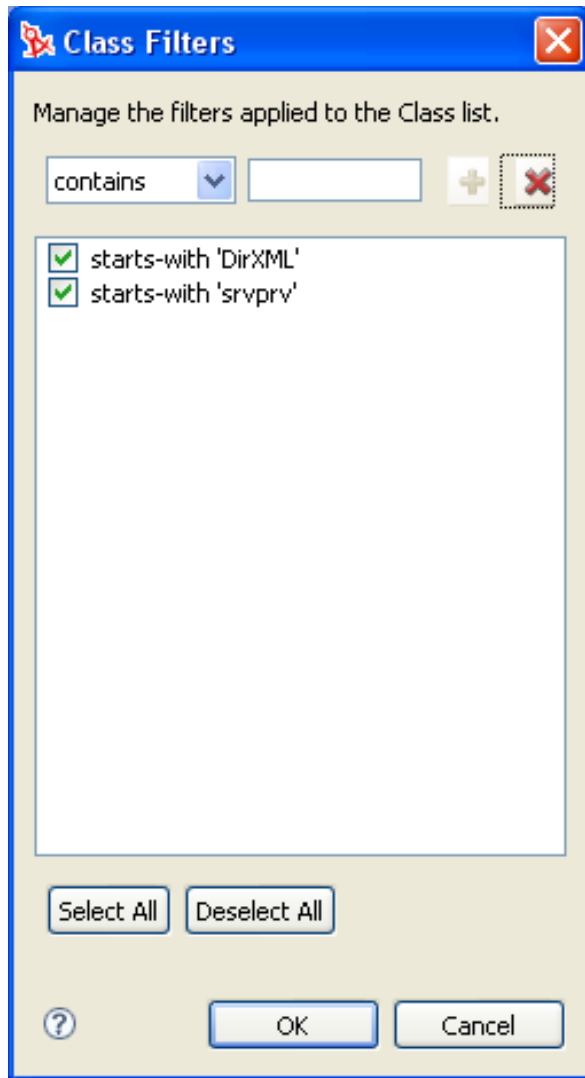
- 5 Click *Finish*.

The property page displays for editing. For more information, see ["Entity Properties"](#) on page 60. You must deploy the entity before it is available to the User Application.

Filter the Object Class List

You can limit the object classes shown in the New Entity dialog by adding a filter. To add a filter:



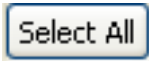
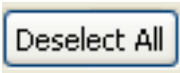
- 1 Click *Configure Filter* to launch the Class List Filters dialog box.



By default, Designer does not apply any class filters. The Class Filter dialog box contains two pre-defined filters (*starts-with "DirXML"* and *starts-with "srvprv"*). To activate them, click *Select All*, then click *OK*. The filters are immediately applied to the object class list. Filters are applied until you deselect them.

- 2 Use the buttons as follows:

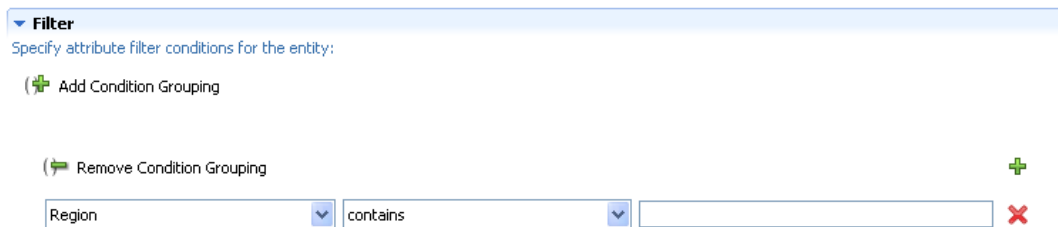
Button	Description
	Choose one of the string comparison operators, such as contains, starts-with, ends-with, then type the string to compare against.

Button	Description
	Adds a filter. Enabled when you define the filter comparison value.
	Removes the selected filter.
	Click this option when you want to use all of the filters. It selects all of the defined filters.
	Click this option when you want to deselect all of the defined filters. If you apply this change, no filters are used.

Adding Entity Filters

You define an entity filter to limit the entries returned for the specified entity. You define the filter based on attributes and their comparison to another value that you specify. For example, you can create a filter so that the User entity includes only those entries whose Region attribute contains Northeast.

- 1 Click *Add Condition Grouping*.



- 2 Use the drop-down list on the left to select an attribute.
- 3 Use the drop-down list in the middle to select a comparison operation.
- 4 Use the entry on the right to specify a value for comparison.
- 5 To specify multiple condition groupings, repeat this procedure. Within a condition grouping, you specify each criterion that you want and connect them by using the logical operations: and, or.

The conditions are evaluated in the order in which you define them.

3.2.3 Adding Attributes

- 1 Select an entity.
- 2 Do any of the following to add an attribute:
 - ♦ Right-click an entity, then select *Add Attribute*.

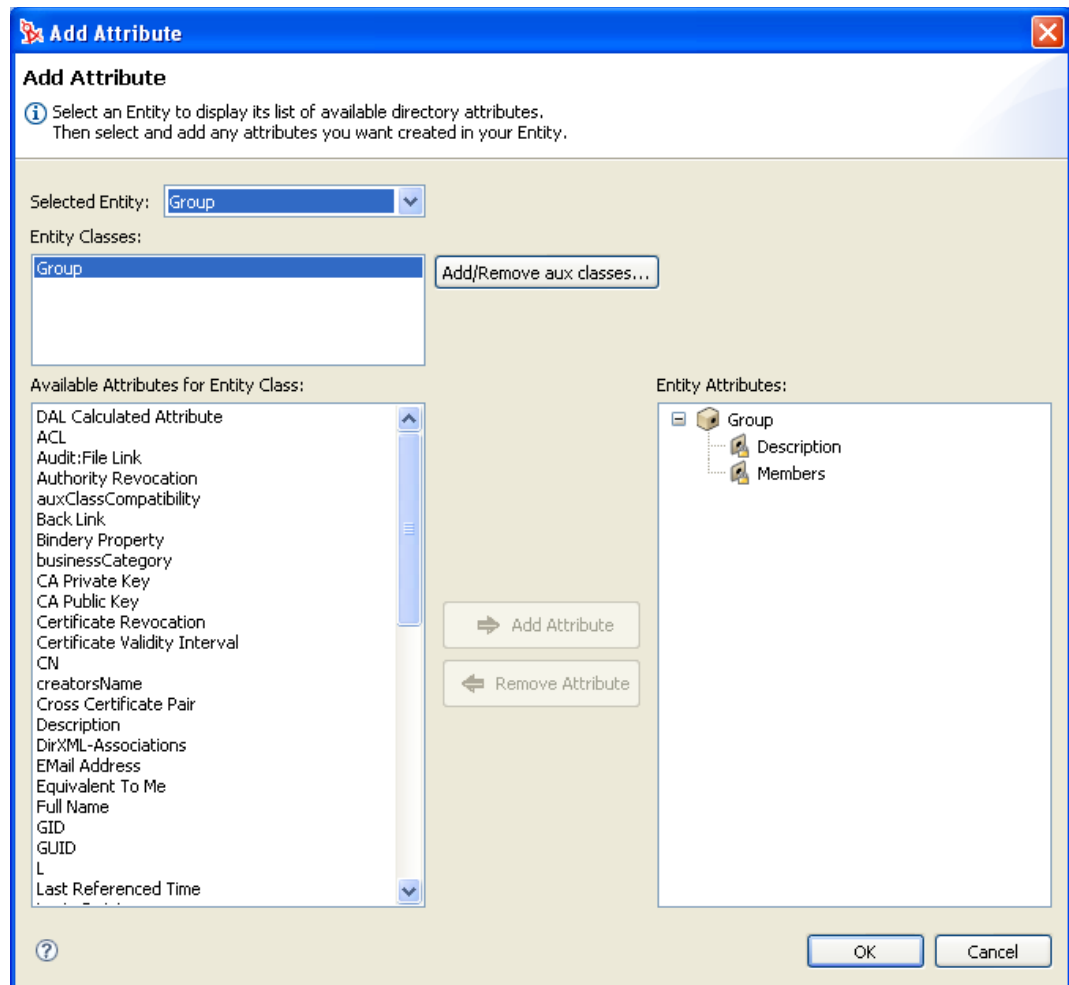
or

- ♦ Click the *Add Attribute* button.

or

- ♦ Click *DAL > New > Attribute*.

You are prompted to choose the entity class that contains the attributes that you want to add to the entity. You can also add (and remove) auxiliary classes if you need to add a class that contains the attribute you are looking for.



- 3 Add attributes by double-clicking them in the *Available Attributes for Entity Class* list.

LDAP operational attributes are supported by the directory abstraction layer editor and User Application; however, when you add an operational attribute, the Edit, Required, and Hidden properties are set to false and are disabled so you cannot change these property values.

TIP: If the attribute you want to add is not displayed in the *Available Attributes from Entity Class* list, you should update Designer's local schema file by following the procedure in [“Updating the Schema Elements List” on page 49](#).

- 4 Click *OK*. The property page displays for editing.

For more information, see [“Attribute Properties” on page 64](#). To make an attribute available to the User Application, you must deploy it.

Adding DAL Calculated Attributes

You can create an attribute that is derived from an expression. For example, you can concatenate two or more attributes to produce a single calculated value. The expressions are ECMAScript compatible and conform to the ECMA 262 Language specification.

Restrictions: Because this attribute type does not map to a specific attribute in the Identity Vault, these attributes cannot be updated, removed, multivalued, required, or searched.

To create a calculated attribute:

- 1 Add an attribute as instructed in [Section 3.2.3, “Adding Attributes,” on page 47](#) and make sure to select *DAL Calculated Attribute* from the *Available Attributes for Entity Class* list.

Designer adds the Attribute with the following restrictions:

Table 3-3 *Calculated Attribute Properties*

Property Name	Description
Expression	Click <i>Build ECMAScript Expression</i> to launch the ECMA Expression Builder. To learn more about how to use the ECMA Expression Builder, see Chapter 10, “Working with ECMA Expressions,” on page 249 .

3.2.4 Updating the Schema Elements List

- 1 With the Identity Manager project open, right-click your Identity Vault, then select *Live > Import Schema*.
- 2 Choose *Import from eDirectory™* and provide the specifications for the eDirectory host.
- 3 Click *Next*.
- 4 Select the classes and attributes to import, then click *Finish*.

3.3 Working with Lists

The lists node lets you define the contents of global lists. You can then define an attribute control type as a global list. When the User Application displays the attribute for editing, the contents of the global list are displayed in a drop-down list for the user to make a selection. By default, the directory abstraction layer includes the global lists described in [Table 3-4](#).

Table 3-4 *Directory Abstraction Layer Default Global Lists*

List Name	Description
Delegate Relationship	Defines the relationships that can be selected when making a Delegate Assignment by relationship. The contents of this list display in a drop-down list box. The values can only be DN attributes from the User entity.

List Name	Description
Email Notification Types	Represents the type of e-mail notification that a user wants to receive when involved in proxy/delegate processing of resource requests. Types are locked.
	WARNING: Do not edit these values.
	This is used by the Preferred Notification attribute of the user entity.
Provisioning Category	Defines the set of categories that organize provisioned resources (entitlements) and provisioning requests. The categories in this list display in: <ul style="list-style-type: none"> ◆ Designer: Provisioning request definition editor plug-in ◆ iManager: Provisioning Request Configuration plug-in ◆ User Application: <i>Requests and Approvals</i> tab

NOTE: You cannot delete these lists or change the key values for the lists. Except for the Email Notification types, you can add and remove items and change existing values and labels.

To create a new global list:

- 1 Launch the New List Wizard in one of these ways:

From Designer's menus:

- ◆ Select *File > New > Provisioning*, select *Directory Abstraction Layer List*, then click *Next*.

When launched from the File menu, the dialog box contains fields not displayed when launched in other ways.

- ◆ Select *DAL > New > List*.

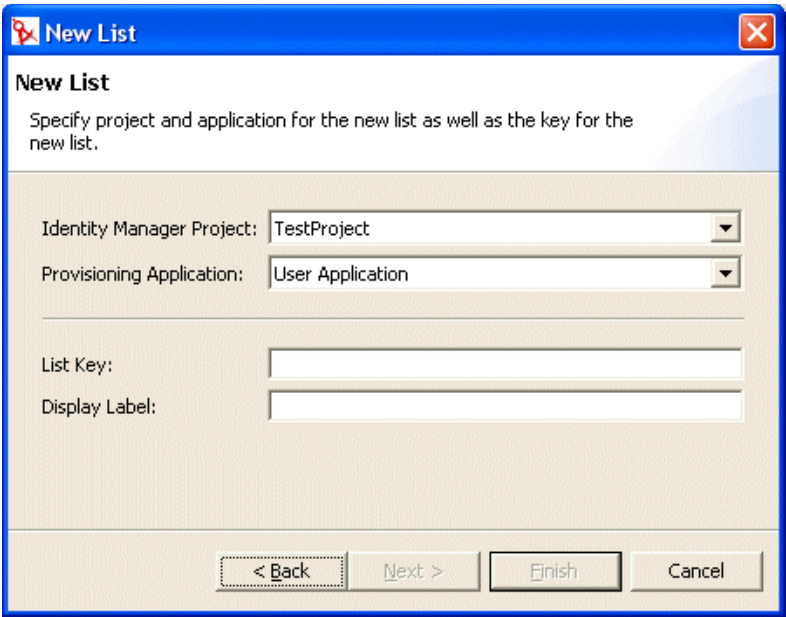
From the Provisioning view:

- ◆ Right-click the Lists node, then select *New*.

From the directory abstraction layer editor:

- ◆ Click *New List*.
- ◆ Right-click the Lists node, then select *Add List*.

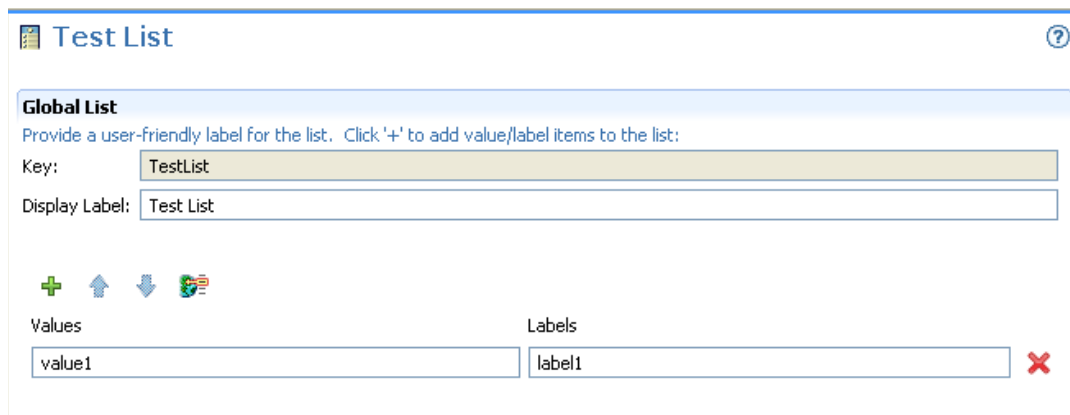
The New List dialog box displays.



2 Fill in the fields as follows:

Field	Description
<i>Identity Manager Project and Provisioning Application</i>	Select the Identity Manager project and provisioning application where you want to add the list. NOTE: These fields display when you launch the wizard from the <i>File</i> menu.
<i>List Key</i>	The unique identifier for the list.
<i>Display Label</i>	The string used when the list is displayed in the User Application. You can localize this label. For more information, see Section 2.9, “Localizing Display Labels,” on page 31.






- 3 Click *Finish*. The Global Lists property page displays for editing.



- 4 Fill in the fields as follows:

Field	Description
<i>Display Label</i>	The name of the list. This is the name displayed in Designer.
<i>Labels</i>	The text for the list item to display in the User Application.
<i>Values</i>	The list item value stored in the Identity Vault. Valid characters include letters, numbers, and the underscore (_) character.

The following table describes the wizard's buttons:

Button	Description
	Adds a new Value
	Moves the row up or down in the list. This order specifies how the labels are displayed in the User Application.
	
	Displays the localization dialog box. For more information on using the dialog box, see Section 2.9, "Localizing Display Labels," on page 31 .
	Deletes the row.

- 5 Save the project.
- 6 Deploy to make it available to the User Application.

3.4 Working with Queries

The Queries node allows you to define commonly used LDAP searches that you can execute from a request or approval form using the DNQuery control or by calling the `globalQuery()` method. To define the query, you specify the directory abstraction layer entity, the search root, the number of rows to retrieve, and the conditions for retrieving the source entity. You can hard-code the conditions (for example, Where LastName contains s) or specify one or more parameters that are supplied by the user on the request or approval form.

To create a query:

- 1 Launch the New Query Wizard in any of these ways:

From Designer's menus:

- ♦ Select *File > New > Provisioning*. Choose *Directory Abstraction Layer Query*, then click *Next*.
- ♦ Select *DAL > New > Query*.

From the Provisioning view:

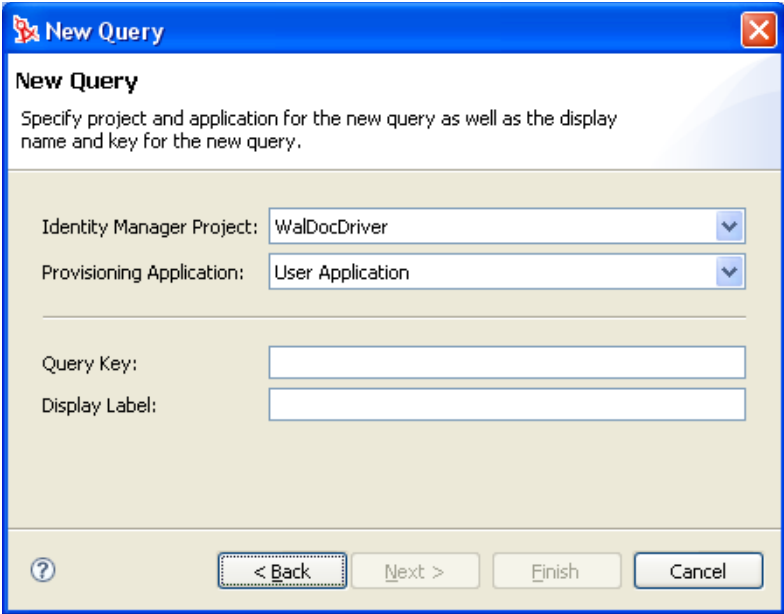
- ♦ Right-click *Query*, then select *Add*.

From the directory abstraction layer editor:

- ♦ Click the *Add Query* button.
- ♦ Right-click *Query*, then select *Add Query*.

The New Query dialog box displays.

NOTE: When launched from the *File* menu, the dialog box contains fields not displayed when launched in other ways.



The image shows a Windows-style dialog box titled "New Query". The title bar is blue with a red "X" button on the right. Below the title bar, the text "New Query" is displayed in bold. Underneath, a smaller text says "Specify project and application for the new query as well as the display name and key for the new query." The main area of the dialog is light beige. It contains two dropdown menus: "Identity Manager Project:" with "WalDocDriver" selected, and "Provisioning Application:" with "User Application" selected. Below these are two empty text input fields labeled "Query Key:" and "Display Label:". At the bottom, there is a row of four buttons: a help button (question mark icon), "< Back", "Next >", and "Cancel". The "Next >" button is disabled.

- 2 Fill in the fields as follows:

Field	What to do
<i>Identity Manager Project and Provisioning Application</i>	Select the correct Identity Manager project and Provisioning Application. NOTE: This field displays when you create queries from the <i>File</i> menu.
<i>Query Key</i>	Type a unique value for the query key. This value is used in the Expression Builder to identify the query.
<i>Display Label</i>	Type a string to display in the directory abstraction layer editor and Provisioning view. This value is not visible in the Expression Builder.

3 Click *Finish*.

The editor creates the query and opens the property page for editing.

4 Select a *Query Entity*. If the entity you want to use is not displayed make sure it is defined in the Entities node.

5 In the *Parameters* section, define one or more parameters for the query. To add parameters:

5a Click *Add Row*.

The screenshot shows a configuration panel titled "Parameters" with a sub-header "Define parameter references:". Below this, there are three icons: a green plus sign, a blue up arrow, and a blue down arrow. The panel is divided into two columns: "Parameter Keys" and "Parameter Display Labels". In the "Parameter Keys" column, there is a text input field containing "param1". In the "Parameter Display Labels" column, there is a text input field containing "label1". To the right of the "Parameter Display Labels" column, there is a red "X" icon.

5b Specify a unique key and a display label for the parameter. You pass this key when calling the `globalQuery()` method on a form. For more information on `globalQuery()`, see [“`globalQuery\(fieldname, key, param\)`” on page 268](#).

5c Add additional parameters by repeating these steps.

6 To further refine the query, add *Query Conditions*.

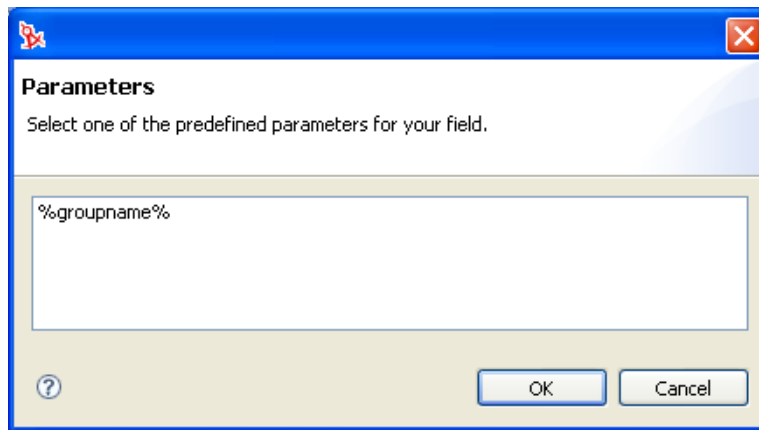
6a Click *Add Condition Grouping* (a *Query Entity* must be selected to enable *Add Condition Grouping*).

The screenshot shows a configuration panel titled "Query Conditions" with a sub-header "Provide conditions for the query entity selected above:". Below this, there is a button labeled "Add Condition Grouping" with a green plus icon. Below the button, there is a "Remove Condition Grouping" button with a green minus icon. At the bottom, there are two drop-down menus, both showing "<Select an item>". To the right of the second drop-down menu, there is a text input field and a red "X" icon.

6b Use the drop-down list on the left to select an attribute. The attributes in this drop-down are the attributes on the selected *Query Entity*.

6c Use the drop-down in the middle to select a comparison operation to perform against your chosen attribute.

- 6d** Use the entry on the right to specify a value to compare against your chosen attribute. You can select a variable name by clicking *Predefined Parameters* to launch the Predefined Parameters dialog.



If the query needs to filter on more than one attribute or condition and you want to control the order in which the conditions are evaluated, you can define multiple conditions or condition groups. Within a condition grouping, you specify each criterion that you want and connect them by using the logical operations: and, or.

- 7** To specify multiple condition groupings, click *Add Condition Groupings* and make your selections from the drop-down list boxes.
- 8** Define the query's LDAP Search properties if you want to narrow the search further than already defined for the selected entity. The query's Search Root, unlike the entity search root, does not support the use of predefined parameters. For more information, see [Section 3.7.3, "Queries Properties," on page 73](#).
- 9** Click *Save*.
- 10** Deploy the query to make it available to the User Application.

3.5 Working with Relationships

The Relationships node allows you to define relationships between entities defined in the directory abstraction layer. The relationships you define are used in the User Application by the Organization Chart and in iManager for defining the team members within a team.

The relationship you define can be between like entities (such as user/user) or unlike entities (such as user/device). You can define conditions for the relationship to further refine it. For example, you might want to create a condition that shows all Manager-Employee relationships and then refine it to show only employees in one particular region, or show all the subordinates of a vice president located in the eastern region.

The following relationships are defined, by default, for the User Application:

- ♦ Group's membership (Org Chart only)
- ♦ Manager-Employee (Org Chart and Team Management)
- ♦ User groups (Org Chart only)

A relationship can only be used by Team Management when the Source and Target entities are both related to the InetOrgPerson object.

To successfully deploy a relationship, all of the components (entities and attributes) of the relationship must already be deployed.

1 YCreate a new relationship in any of these ways:

From Designer's menus:

- ♦ Select *File > New > Provisioning*. Choose *Directory Abstraction Layer Relationship*, then click *Next*.
- ♦ Select *DAL > New > Relationship*.

From the Provisioning view:

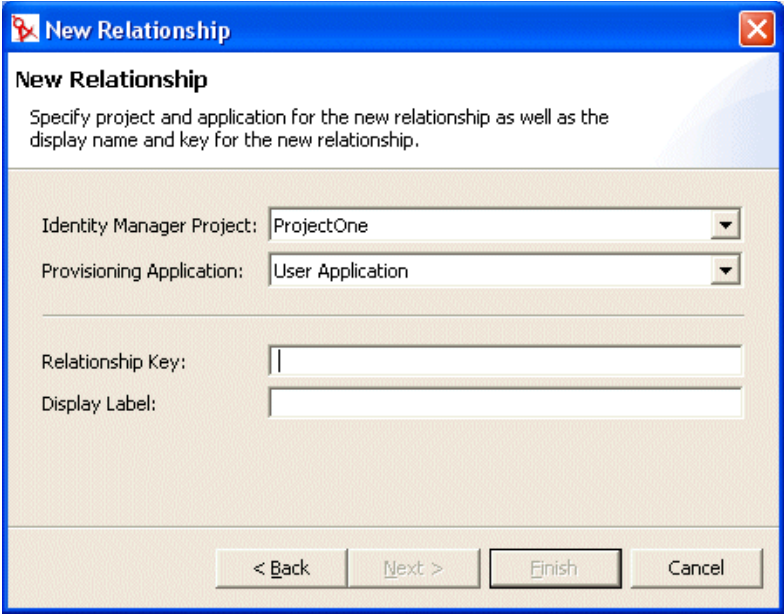
- ♦ Right-click *Relationships*, then select *Add*.

From the directory abstraction layer editor:

- ♦ Click the *Add Relationship* button.
- ♦ Right-click *Relationships*, then select *Add Relationship*.

The New Relationship dialog box displays.

NOTE: When launched from the *File* menu, the dialog box contains fields not displayed when launched in other ways.

The image shows a Windows-style dialog box titled "New Relationship". It has a blue title bar with a red "X" button in the top right corner. The main area has a light beige background. At the top, there's a section header "New Relationship" followed by a descriptive text: "Specify project and application for the new relationship as well as the display name and key for the new relationship." Below this, there are four input fields: "Identity Manager Project:" with a dropdown menu showing "ProjectOne", "Provisioning Application:" with a dropdown menu showing "User Application", "Relationship Key:" with a text box containing a single vertical bar "|", and "Display Label:" with an empty text box. At the bottom, there are four buttons: "< Back", "Next >", "Finish", and "Cancel".

2 Fill in the fields as follows:

Field	What to do
<i>Identity Manager Project and Provisioning Application</i>	Select the correct Identity Manager project and Provisioning Application. NOTE: This field displays when you create relationships from the <i>File</i> menu.
<i>Relationship Key</i>	Type a unique value for the relationship key.
<i>Display Label</i>	Type the string to display when the relationship displays in the User Application.

3 Click *Finish*.

The editor creates the relationship and opens the property page for editing.

The screenshot shows the Identity Manager configuration interface. On the left is a tree view of the configuration hierarchy: Entities (Group, Description, Members, Manager Lookup, User, User Lookup), Lists (Delegatee Relationship, Email notification types, Preferred Locale, Provisioning Category, Request Scope), Queries, and Relationships (Group's membership, Manager-Employee, User groups, Configuration). The 'Group's membership' relationship is selected. The main panel on the right is titled 'Group's membership' and contains the following sections:

- Access:** 'Specify the type of application which can use this relationship:' with checkboxes for 'Used by Organizational Chart' (checked) and 'Used by Team-Management' (unchecked).
- Relationship:** 'Provide a user-friendly label for the relationship:' with a 'Key' field containing 'group2users' and a 'Display Label' field containing 'Group's membership'.
- Source Object:** 'Source Entity' dropdown set to 'Group' and 'Source Attribute' dropdown set to 'This entity's key'.
- Target Object:** 'Target Entity' dropdown set to 'User' and 'Target Attribute' dropdown set to 'Group'.
- Conditions:** A section with the text 'Specify any additional conditions for your target attribute:' and a green plus icon to add conditions.

For property definitions, see [Section 3.7.4, “Relationship Properties,”](#) on page 74.

To delete a relationship:

- 1 Right-click the relationship you want to delete, then click *Delete*.

To add a relationship condition:

- 1 Click *Add Row*.
- 2 Use the drop-down list box (on the left) to select an attribute. The attributes in this drop-down are attributes on the *Target Object* entity.
- 3 Select an operator from the middle drop-down list box.

- 4 Use the text field on the right to specify the comparison value to complete the condition. For example:

The screenshot shows the configuration interface for a relationship named "Manager-Employee(region)". The interface is divided into several sections:

- Access:** Contains checkboxes for "Used by Organizational Chart" and "Used by Team-Management", both of which are checked. There is also an unchecked checkbox for "Enable Cascading Relationship" and a spinner control for "Maximum levels to cascade" set to 0.
- Relationship:** Contains a "Key" field with the value "user2Users2" and a "Display Label" field with the value "Manager-Employee(region)".
- Source Object:** Contains a "Source Entity" dropdown set to "User" and a "Source Attribute" dropdown set to "This entity's key".
- Target Object:** Contains a "Target Entity" dropdown set to "User" and a "Target Attribute" dropdown set to "Manager".
- Conditions:** A section with a plus icon to add conditions. It shows a single condition: "Region" (from a dropdown) "equals" (from a dropdown) "Latin" (in a text field). A red X icon is next to the text field.




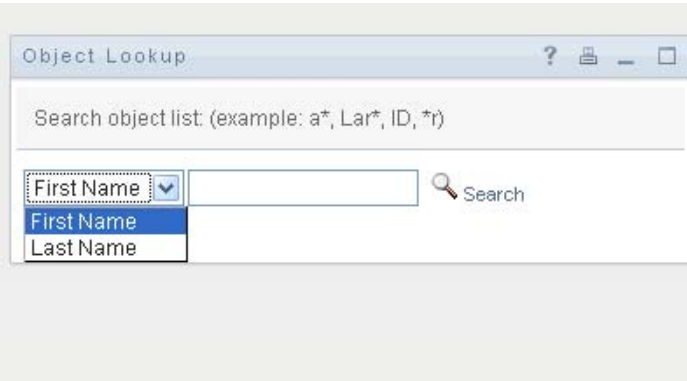
You can create a condition that filters on more than one attribute or condition and connect the attributes by using the logical operations: and, or. The conditions are evaluated in the order in which you define them.

3.6 Working with Configuration Settings

The Configuration node allows you to set general configuration properties for the User Application.

Table 3-5 Configuration Settings

Property	Description
Default 'My Profile' Entity	Defines the entity to display when the user clicks <i>My Profile</i> in the user interface. This field is restricted to show only entities whose object class is user (or LDAP inetOrgPerson).
Default LDAP Naming Attributes	Defines the default LDAP naming attribute if the entity's Create Naming Attribute is not defined.

Property	Description
Default Team Management Attributes	<p>The attributes used to look up team members. In the My Team's Work and My Team's Settings pages of the User Application, the user is able to search for team members by clicking the search icon. For example:</p> <p>Team Member: <input type="text"/>   </p> <p>The search displays the attributes specified here, for example:</p>  <p>These settings only affect the lookups performed by Team Managers. The User Application administrator will always only see First Name and Last Name.</p>
Container Classes	<p>This provides the <i>Create User or Group</i> action with the contents of a selection list of container classes. The user selects a container from the selection list as the location for the newly created object.</p>

3.7 Directory Abstraction Layer Property Reference

The section provides definitions for the properties for the following abstraction layer nodes:

- ♦ [Section 3.7.1, “Entity Properties,” on page 60](#)
- ♦ [Section 3.7.2, “Attribute Properties,” on page 64](#)
- ♦ [Section 3.7.3, “Queries Properties,” on page 73](#)
- ♦ [Section 3.7.4, “Relationship Properties,” on page 74](#)

3.7.1 Entity Properties

You can set the following kinds of properties on entities:

- ♦ [“Entity Access Properties” on page 61](#)
- ♦ [“Entity General Properties” on page 61](#)
- ♦ [“Entity Auxiliary Properties” on page 61](#)
- ♦ [“Entity Search Properties” on page 62](#)

- ♦ “Entity Create Properties” on page 63
- ♦ “Entity Password Management Properties” on page 63
- ♦ “Using Predefined Parameters” on page 63

Entity Access Properties

Access Properties control how the User Application interacts with the entity.

NOTE: You can also access the access properties by selecting *DAL > Set Global Access*.

Table 3-6 *Entity Access Properties*

Property Name	Description
Create	When selected, this object can be created by the User Application.
Edit	When deselected, this object cannot be changed by the User Application regardless of the underlying ACLs. When selected, this object is editable, but the Identity Vault ACLs are used to determine this.
View	When selected, this object can be displayed by the User Application.
Remove	When selected, this object can be deleted by the User Application.

Entity General Properties

Table 3-7 *Entity General Properties*

Property Name	Description
Key	The unique identifier for this entity. It defines the way the User Application references this object. It is defined when the entity is created and cannot be modified after the entity is created.
Display Label	Defines how the object is shown in the user interface.
Class Name	The eDirectory object class name.
LDAP Name	The LDAP object class name.
Include in Search	When selected, this entity is searchable in the User Application. Entities used in queries by identity portlets (such as Entity Search List or Entity Org Chart) must be selected (true).

Entity Auxiliary Properties

Table 3-8 *Entity Auxiliary Properties*

Property Name	Description
Auxiliary Classes	<p>A list of zero or more auxiliary classes for this entity. If adding auxiliary classes, you are prompted to define:</p> <ul style="list-style-type: none">♦ The auxiliary class by selecting from the list of those available♦ Whether it is searchable♦ Whether to <i>Add Always</i>. When True (selected), the object class is automatically added when the entity is modified in the User Application. Modification includes create or update operations. When False, the object class is only added if an attribute associated with the auxiliary class is modified.

Entity Search Properties

Table 3-9 *Entity Search Properties*

Property Name	Description
Search Container	<p>The distinguished name of the LDAP node or container where searching starts (the search root). For example: <code>ou=sample,o=ourOrg</code></p> <p>You can browse the Identity Vault to select the container, or you can use one of the predefined parameters described in “Using Predefined Parameters” on page 63.</p>
Search Scope	<p>Specifies where the search occurs in relation to the search root. Values are:</p> <p><Default>: This search scope is the same as selecting <i>Containers and subcontainers</i>.</p> <p>Container: The search occurs in the search root DN and all entries at the search root level.</p> <p>Container and subcontainers: The search occurs in the search root DN and all subcontainers. This is the same as selecting <i><Default></i>.</p> <p>Object: Limits the search to the object specified. This search is used to verify the existence of the specified object.</p>
Search Time Limit [ms]	Specify a value in milliseconds or specify 0 for no time limit.
Max Search Entries	Specify the maximum number of search result entries you want returned for a search. Specify 0 if you want to use the runtime setting. Recommendations: Set it between 100 and 200 for greatest efficiency. Do not set it over 1000.

Property Name	Description
Perform Automatic Query	<p>When selected, performs an automatic query of the entity and presents the results in a selectable list. Do not choose this option if the data returned will be a large number because it will force the user to scroll through a large result set.</p> <p>When not selected, allows the user to specify the search criteria for the entity query, then presents the results in a selectable list.</p>

Entity Create Properties

Table 3-10 *Entity Create Properties*

Property Name	Definition
Create Container	<p>The name of the container where a new entity of this type is created.</p> <p>You can browse the Identity Vault to select the container, or you can use one of the predefined parameters described in “Using Predefined Parameters” on page 63.</p> <p>If you do not specify this value, then the Create portlet prompts the user to specify a container for the new object. The portlet uses the search root specified in the entity definition as the base and allows the user to drill down from there. If there is no search root specified in the entity definition then it uses the root DN specified during the User Application installation.</p>
Create Naming Attribute	The naming attribute of the entity. It is the relative distinguished name (RDN). This value is only necessary for entities where the access parameter Create is selected.
LDAP attribute	The LDAP attribute for the Create Naming Attribute .
Create Naming Label	Display label displayed in the User Application for the Create Naming Attribute .

Entity Password Management Properties

Table 3-11 *Entity Password Management Properties*

Property Name	Definition
Password required when entity is created	If the password attribute is required, set this value to True (selected) to ensure that one is required by the Create portlet. If a password is required, then you cannot create this entity in a workflow.

Using Predefined Parameters

The directory abstraction layer editor allows you to use predefined parameters for certain values.

Table 3-12 *Predefined Parameters*

Predefined Parameter	Description
%driver-root%	Represents the Provisioning Driver DN. This value is specified during the User Application configuration during installation or a later configuration. It is stored in the User Application's realm configuration.
%user-root%	Represents the User Container DN. This value is specified during the User Application configuration during installation or a later configuration. It is stored in the User Application's realm configuration.
%group-root%	Represents the Group Container DN. This value is specified during the User Application configuration during installation or a later configuration. It is stored in the User Application's realm configuration.

3.7.2 Attribute Properties

You can set the following kinds of properties on attributes:

- ♦ “Attribute Access Properties” on page 64
- ♦ “Attribute General Properties” on page 65
- ♦ “Attribute Default Value Properties” on page 66
- ♦ “Attribute UI Control Properties” on page 66
- ♦ “Attributes and DNLookup Properties” on page 68

Attribute Access Properties

NOTE: You can set attribute access for all of an entity's attributes by selecting *DAL > Set Attribute Access*, right-clicking an entity and selecting *Set Attribute Access*.

Table 3-13 *Attribute Access Properties*

Name	Description
Edit	When selected, this attribute can be edited/modified by the User Application. Even if it is selected (True), the attribute might still not be editable if the underlying Identity Vault ACLs/effective rights prevent it.
Enable	When deselected, this attribute cannot be used by the User Application. It is the same as removing the entry from the file.

Name	Description
Hide	<p>Controls whether the <i>Hide</i> check box in the User Application is enabled or disabled. The <i>Hide</i> check box allows users to control whether an attribute (such as a photo) is displayed by the application.</p> <p>When deselected, the <i>Hide</i> check box is disabled for this attribute, so the user cannot choose to hide this attribute.</p> <p>When selected, the <i>Hide</i> check box can be enabled in the User Application. However, the following must also be true of the logged-in user.</p> <ul style="list-style-type: none"> ♦ He or she is either the owner of the attribute or a User Application Administrator. ♦ He or she has Trustee rights to update the <code>srvprvHideAttributes</code> attribute on the Identity Vault. <p>If these requirements are not met, then the Hide check box is disabled in the user interface even if this setting is selected (True).</p> <hr/> <p>TIP: When a user hides an attribute that contains an image, users who have viewed the image might continue to see it until their browser cache is refreshed.</p>
Multivalue	<p>Specifies whether this attribute can be multivalued, for example, a phone number.</p> <p>When selected, the attribute can be multivalued.</p>
Read	<p>When this option is selected, the User Application can query this attribute. For most attributes this should be selected (true), but for some attributes, like password, it should be deselected.</p>
Require	<p>When this option is selected, the attribute must be supplied.</p>
Search	<p>When this option is selected, the User Application can search on this attribute. Attributes that are used in queries by identity portlets (such as Entity Search List or Entity Org Chart) or request and approval forms must be selected.</p> <hr/> <p>TIP: If an attribute used in a search is also indexed in eDirectory, the search is faster.</p>
View	<p>When this option is selected, the User Application can display this attribute. In most cases this is selected, but for attributes like password, it should be deselected. If you specify it in a request or approval form, view must be selected.</p>

Attribute General Properties

Table 3-14 *Attribute General Properties*

Property Name	Description
Key	The unique identifier for the attribute.
Display Label	The label that is displayed in the User Application.
Attribute Name	The eDirectory name for this attribute.
LDAP Name	The LDAP name for this attribute.

Attribute Default Value Properties

This value is used when an object is created via the Create identity portlet or through a workflow. You can express the default value as a literal or an ECMAScript expression. You cannot use a default value as part of a calculated attribute. If defined as an ECMAScript expression, it is resolved at runtime. If you define both the literal and an expression, the expression takes precedence.

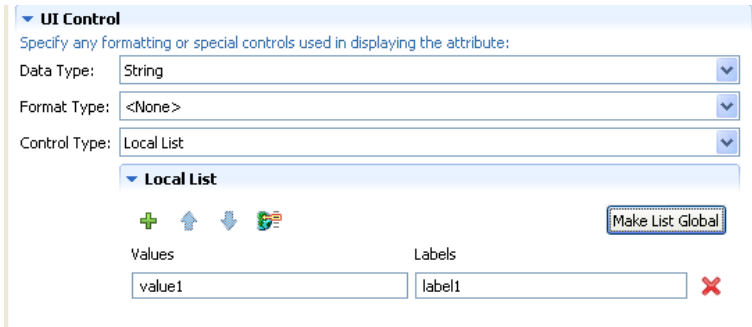
TIP: If you want the default value to be displayed by the Create portlet, you must define the access property viewable as true (selected). If you want the user to be able to change the value, you must set the editable property to true.

Attribute UI Control Properties

Table 3-15 *Attribute UI Control Properties*

Property Name	Description
Data Type	Choose a data type from the following list: <ul style="list-style-type: none">♦ Binary♦ Boolean♦ DN♦ Integer♦ LocalizedString♦ String♦ Time
Format Type	Used by the User Application to format data. Format types include: <ul style="list-style-type: none">♦ None♦ AOL IM♦ Email♦ Groupwise IM♦ Image♦ Phone Number♦ Yahoo IM♦ Image URL♦ Date♦ DateTime

The Format Types are dependent on the data type. For example, a Time data type can only be associated with Date and DateTime formats.

Property Name	Description
Control Type	<p>Types include:</p> <p><i>DNLookup</i>: Defines that this attribute contains a DN reference. Use when you want to:</p> <ul style="list-style-type: none"> ♦ Populate a list with the results of a DN search among related entities. ♦ Maintain referential integrity across DN referenced attributes during updates and deletes. ♦ Use the attribute in an object selector dialog box. Object selectors are used by certain identity portlets, such as Detail, and are also available to the form controls you can define for provisioning request and approval forms. <p>The User Application uses this information to generate special user interface elements (such as an object selector), and to perform optimized searches based on the DNLookup definition.</p> <p>For more information on defining this property, see the “Attributes and DNLookup Properties” on page 68. For more information on the object selector dialog box for request and approval forms, see Section 6.6.2, “Working with Object Selectors,” on page 148.</p> <p><i>Global List</i>: Display this attribute as a drop-down list whose contents are defined in a file outside of this attribute definition. Click <i>Go to list</i> to access the Global List editor for the selected list.</p> <p>For more information, see Section 3.3, “Working with Lists,” on page 49.</p> <p><i>Local List</i>: Display this attribute as a drop-down list whose contents are defined with this attribute. To define a local list:</p> <ol style="list-style-type: none"> 1. With the attribute selected, set the control type to <i>Local List</i>.  <ol style="list-style-type: none"> 2. Use the buttons to add or remove list items. Use the up-arrow and down-arrow buttons to change the position of the item in the list. <p>In the <i>Value</i> column, type the value to write to the Identity Vault. It can include letters, numbers, and underscore (_) character.</p> <ol style="list-style-type: none"> 3. In the <i>Labels</i> column, type the text you want displayed in the user interface. <p><i>Range</i>: Use the Range control type with Integer data types to restrict user input to a sequential range of values. Define the range's start and end values.</p>

Attributes and DNLookup Properties

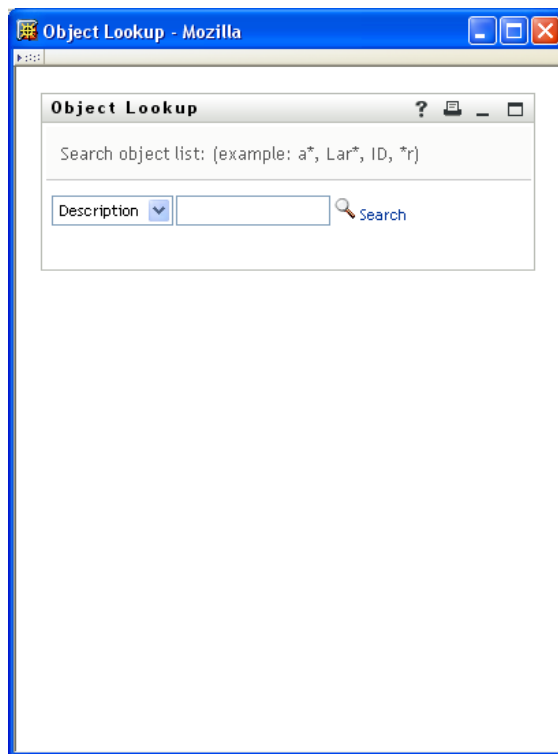
When you define an attribute as a DNLookup control type, it means that:

- ◆ This attribute can be used in an object selector dialog box which allows users to select from a list of possible values when searching on this attribute.
- ◆ When this attribute is created, populated, or deleted through the User Application, an attribute on a related entity is updated appropriately depending on the user action (create, delete, update) to maintain referential integrity.

DNLookups for Object Selectors

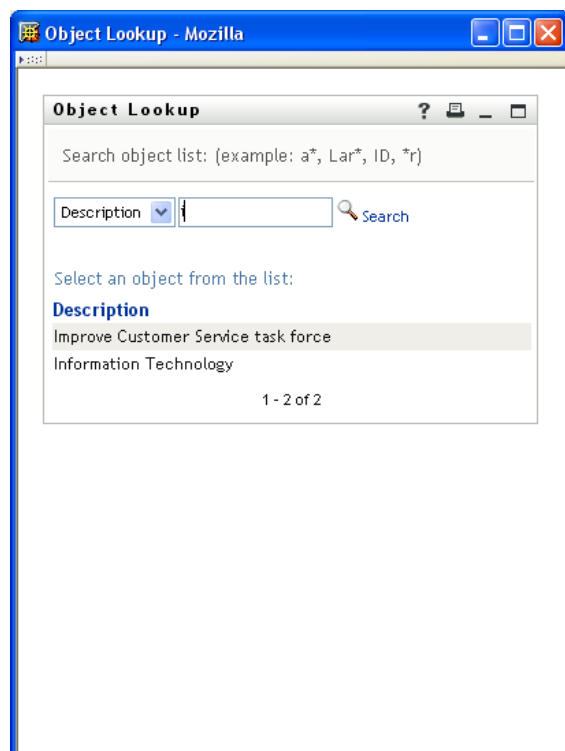
The DNLookup Display properties for a particular attribute define the contents of the object selectors in the User Application. Object selectors are displayed by the Identity Self-Service portlets and in workflow request and approval forms. They provide a convenient way for users for users to search and select objects that represent DNs (such as users or groups). The object selector displays a drop-down list of attributes; the user can select one of the attributes and then enter search criteria for that attribute. In this example, the user searches for groups by group description.

Figure 3-2 *Sample Object Selector*



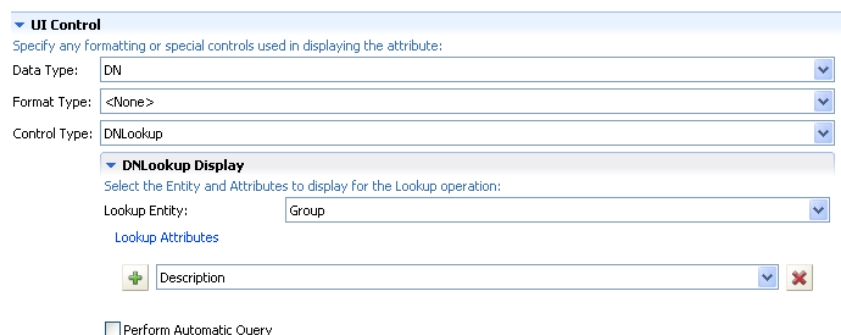
The result of the user's selection looks like this:

Figure 3-3 Sample Object Selector Results



The DNLookup display properties control the contents of the object selector and the result set. The object selector, shown above, displays this way because it was based on the group attribute of the user entity. The group attribute is defined as a DNLookup control type as shown here:

Figure 3-4 Group DNLookup Definition



This definition also controls the way identity portlets provide a selection list of groups for a user. For example, a user might choose to do a Directory Search to find a user in a group, but the group name

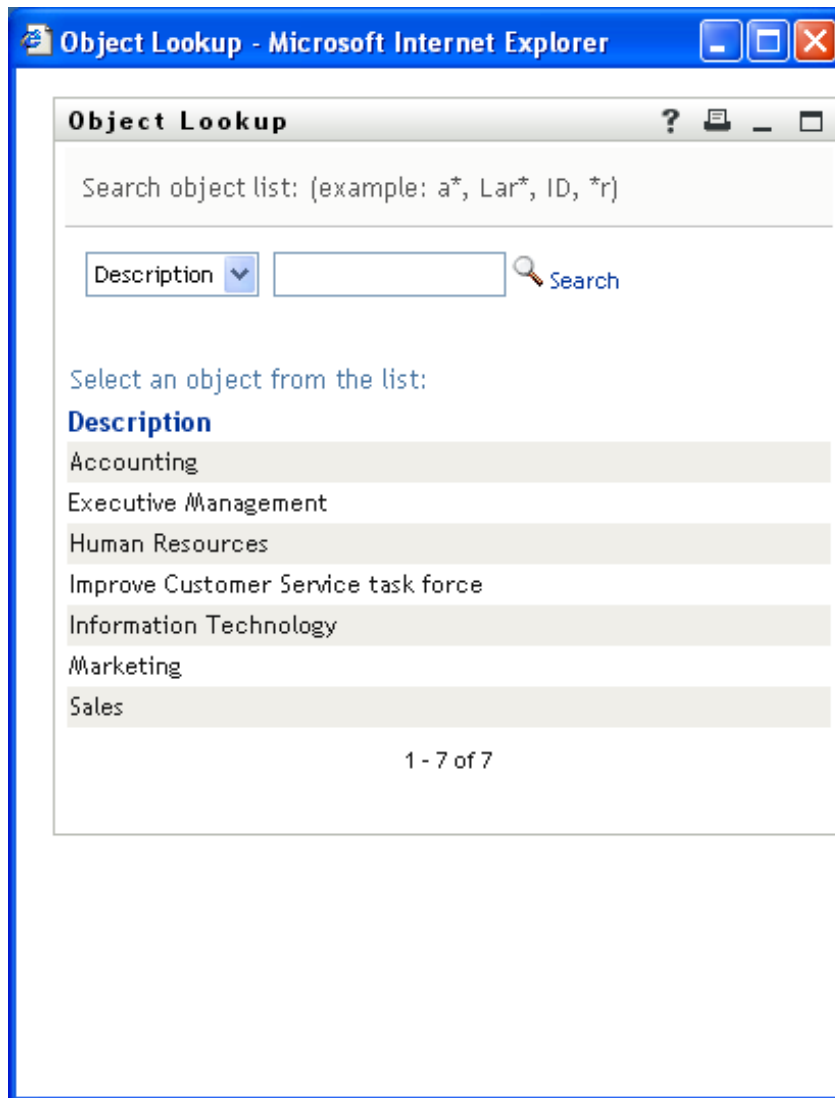
is unknown. The user would select User as the object to search for and select group as the search criteria, as follows:

Figure 3-5 *Search Criteria*

The screenshot shows a window titled "Search List" with standard window controls (help, maximize, close) in the top right corner. Below the title bar is a yellow header area containing the text "Basic Search". The main content area is white and contains the following elements: a label "Search for:" followed by a dropdown menu currently showing "User"; a label "With this criteria:" followed by a row of three dropdown menus. The first dropdown shows "Group", the second shows "equals", and the third is empty. To the right of the third dropdown are three small icons: a magnifying glass, a document, and a checkmark. Below these dropdowns is a "Search" button. At the bottom of the window is a yellow footer area containing two links: "My Saved Searches" with a folder icon and "Advanced Search" with a magnifying glass icon.

Because the members attribute is a DNLookup for the user entity, the *Lookup* icon displays. If the user selects it, then a list of possible groups displays.

Figure 3-6 *Object Lookup*



When the user picks a group, then he or she can select a group from the list and all of the members of that group are displayed.

NOTE: When the Perform Automatic Query property is not selected (False), the object selector is not populated when first displayed to the user and the user must enter selection criteria. The example above illustrates the object selector that displays when the Perform Automatic Query property is selected (True).

DNLookups for Referential Integrity

DNLookups for updates and synchronization are important because LDAP allows group relationships to map in both directions. For example, your data might be set up so that:

- ♦ The User object contains a group attribute. The group attribute is multi-valued and lists all of the groups to which a user belongs.
- ♦ The Group object contains a user attribute. The user attribute is multi-valued and lists all of the users that belong to the group.

This means that you can have an attribute on the user object that shows all the groups a user belongs to, and on the Group object you have a DN attribute that includes all the members of that group.

When the user requests an update, the User Application must honor the relationships and ensure that the target and source attributes are synchronized. In the DNLookup, you specify both attributes that must be synchronized. You can use this technique to provide synchronization between any objects that are related not just group structural objects. Create this kind of DNLookup control type by specifying the advanced DNLookup properties described in the *DNLookup Relational Integrity properties* reference.

DNLookup Property Reference

Table 3-16 *DNLookup Display Properties*

Property Name	Description
Lookup Entity	The name of the entity to search, for example, the User entity contains an attribute for Manager. To populate that field, you'd need to know which users are Managers.
Lookup Attributes	Choose one or more attributes to display when a search is performed.
Perform Automatic Query	Defines how the Lookup Attributes are displayed. <ul style="list-style-type: none">♦ When this option is selected, the form or portlet performs an automatic query of the entity and presents the results in a selectable list. This option is not recommended if a large amount of data can be returned because it forces the user to scroll through a large result set.♦ When this option is deselected, allows the user to specify the search criteria for the entity query, then presents the results in a selectable list.

Table 3-17 *DNLookup Detail Properties*

Property Name	Description
Detail entity	The key of the entity whose details you want displayed if the user requests more information by clicking a hypertext link in the User Application. When you define a DNLookup, the identity portlets are able to provide a hypertext link that allows users to display the details of the linked object.

The DNLookup Relational Integrity properties are used for synchronizing data between two objects such as groups and group members.

Table 3-18 *DNLookup Relational Integrity Properties*

Property Name	Description
Source Attributes to Update	Name of the attribute to update. The attribute must contain a DN reference to the Target Attributes to Update. This is required to synchronize attributes on two different objects.
Target Attributes to Update	Name of the attribute that must be updated along with the Source Attributes to Update. This is an LDAP attribute name. This is required to synchronize attributes on two different objects. The attribute must contain a DN reference.
Target Auxiliary Classes Needed, if any	Name of the auxiliary class that contains the Target Attributes to Update.

3.7.3 Queries Properties

You can set the following kinds of Queries properties:

- ♦ [“Queries General Properties” on page 73](#)
- ♦ [“Query Parameters Properties” on page 73](#)
- ♦ [“Query Search Properties” on page 74](#)

Queries General Properties

Table 3-19 *Queries General Properties*

Property Name	Description
Key	A unique value for the query key. This value is used in the Expression Builder to identify the query. The key is specified at the query creation time. It cannot be modified after the query is created.
Query Entity	Select an entity from the drop-down list box. The resulting LDAP search is on this entity.
Display Label	Type a string to display in the directory abstraction layer editor and Provisioning view. This value is not visible in the Expression Builder.

Query Parameters Properties

Table 3-20 *Queries Parameters Properties*

Property Name	Description
Parameter Keys	A unique identifier for the key. You pass this key when calling the <code>globalQuery()</code> method on a form.

Property Name	Description
<i>Parameter Display Labels</i>	A label to identify the key.

Query Search Properties

If left blank, the query search properties default to the search properties specified for the selected entity. Specify the query search properties to further refine the search scope already defined for the entity. You cannot specify predefined parameters (for example, %user-root%) in the query's search properties.

Table 3-21 *Query Search Properties*

Property Name	Description
Search Root	Specifies the location in the LDAP tree where the LDAP search defined by the query begins. .
Search Scope	<p>Specifies where the search occurs in relation to the search root. Values are:</p> <p><Default>: This search scope is the same as selecting <i>Containers and subcontainers</i>.</p> <p>Container: The search occurs in the search root DN and all entries at the search root level.</p> <p>Container and subcontainers: The search occurs in the search root DN and all subcontainers. This is the same as selecting <i><Default></i>.</p> <p>Object: Limits the search to the object specified. This search is used to verify the existence of the specified object.</p>
Max Search Entries	<p>Specify the maximum number of search result entries you want returned for a search. Specify 0 if you want to use the runtime setting.</p> <p>Recommendations: Set it between 100 and 200 for greatest efficiency. Do not set it over 1000</p>

3.7.4 Relationship Properties

Relationship properties include:

- ♦ [“Relationship Access Properties” on page 74](#)
- ♦ [“Relationship Properties” on page 75](#)

Relationship Access Properties

Table 3-22 *Relationship Access Properties*

Property Name	Description
Used by Organizational Chart	When selected, this relationship can be used by the Org Chart portlet.

Property Name	Description
Used by Team Management	<p>When selected, this relationship can be used to define the provisioning team members in iManager.</p> <p>For example, if <i>Used by Team Management</i> is selected for the manager-employee relationship, then the provisioning application administrator can use this relationship to define the team members as all users that report to the team manager.</p> <p>If <i>Enable Cascading Relationship</i> is selected, then the team can include several levels within the organization. You define the number of levels via <i>Maximum Levels to Cascade</i>.</p>

Relationship Properties

Table 3-23 *Relationship Properties*

Property Name	Description
Key	<p>The read-only unique identifier for the relationship.</p> <p>TIP: You specify this value in the Org Chart Portlet preference sheet.</p>
Display Label	<p>Specify a name to display when this relationship is displayed in the User Application. For example, this value is displayed when users click Choose Org Chart from the Detail portlet.</p> <p>Click Localize to provide the translation for the display label text.</p>
Source Entity	<p>Choose an entity from the drop-down list.</p> <p>The entity that you choose becomes the parent or source object in the organization chart hierarchy. In a Manager-Employee relationship, the Source Entity is User. For a Group-Member relationship, the source entity is Group.</p> <p>Directory abstraction layer requirements: The entities in this list are a subset of the entities defined in the directory abstraction layer. Source entities must have the view access property selected (true).</p>
Source Attribute	<p>Choose an attribute from the drop-down list.</p> <p>This attribute is used to find matching target entities. When the value of this attribute matches a corresponding value on an attribute of the target entity (see Target Attribute below), then a relationship can be established.</p> <p>Directory abstraction layer requirements: This list of attributes is populated using the selected Source Entity's attributes. It includes any attributes that are searchable and readable.</p>
Target Entity	<p>Choose an entity for the child object in the hierarchy. In a Manager-Employee relationship, it is user.</p> <p>This entity must contain the attribute that is related to the Source attribute.</p>

Property Name	Description
<i>Target Attribute</i>	<p>Choose the attribute that matches the Source Attribute.</p> <p>This is the target entity's attribute used to find matching source entities. When the value of this attribute matches a corresponding value on the source entity (see Target Attribute above), then a relationship can be established.</p>
<p>NOTE: The Org Chart portlet does not fully support dynamic groups; you cannot define a dynamic group as the Source entity, but you can define a dynamic group as the target entity.</p>	

Working with the Provisioning Request Definition Editor

4

This section provides general guidelines for using the provisioning request definition editor. Topics include:

- ♦ [Section 4.1, “About the Provisioning Request Definition Editor,” on page 77](#)
- ♦ [Section 4.2, “Basic Steps for Creating a Provisioning Request Definition,” on page 83](#)
- ♦ [Section 4.3, “Guidelines for Creating Workflows,” on page 84](#)
- ♦ [Section 4.4, “Working with the Installed Templates,” on page 90](#)
- ♦ [Section 4.5, “Debugging a Workflow,” on page 92](#)

4.1 About the Provisioning Request Definition Editor

The provisioning request definition editor allows you to create custom provisioning request definitions by using a rich set of Eclipse-based design tools. The provisioning request definition editor lets you define the basic characteristics of the provisioning request, design the associated workflow, and model the initial request and approval forms.

Identity Manager ships with a set of provisioning request *templates* that you can use to create your definitions. The templates model some common workflow design patterns. However, if you want complete control over the behavior of your workflows, you can create your provisioning request definitions from scratch.

NOTE: For details on using the templates, see [Section 4.4, “Working with the Installed Templates,” on page 90](#).

4.1.1 How the Provisioning Request Definition Editor Fits into the Identity Manager Architecture

A key feature of Identity Manager is *workflow-based provisioning*, which is the process of managing user access to secure resources in an organization. These resources can include digital entities such as user accounts, computers, and databases. Provisioned resources are mapped to Identity Manager entitlements or to entities in the directory abstraction layer.

Identity Manager can service a wide range of *provisioning requests*. Provisioning requests are user or system actions intended to grant or revoke access to organizational resources. They can be initiated directly by the end user through the Identity Manager User Application, or indirectly in response to events occurring in the Identity Vault (eDirectory™).

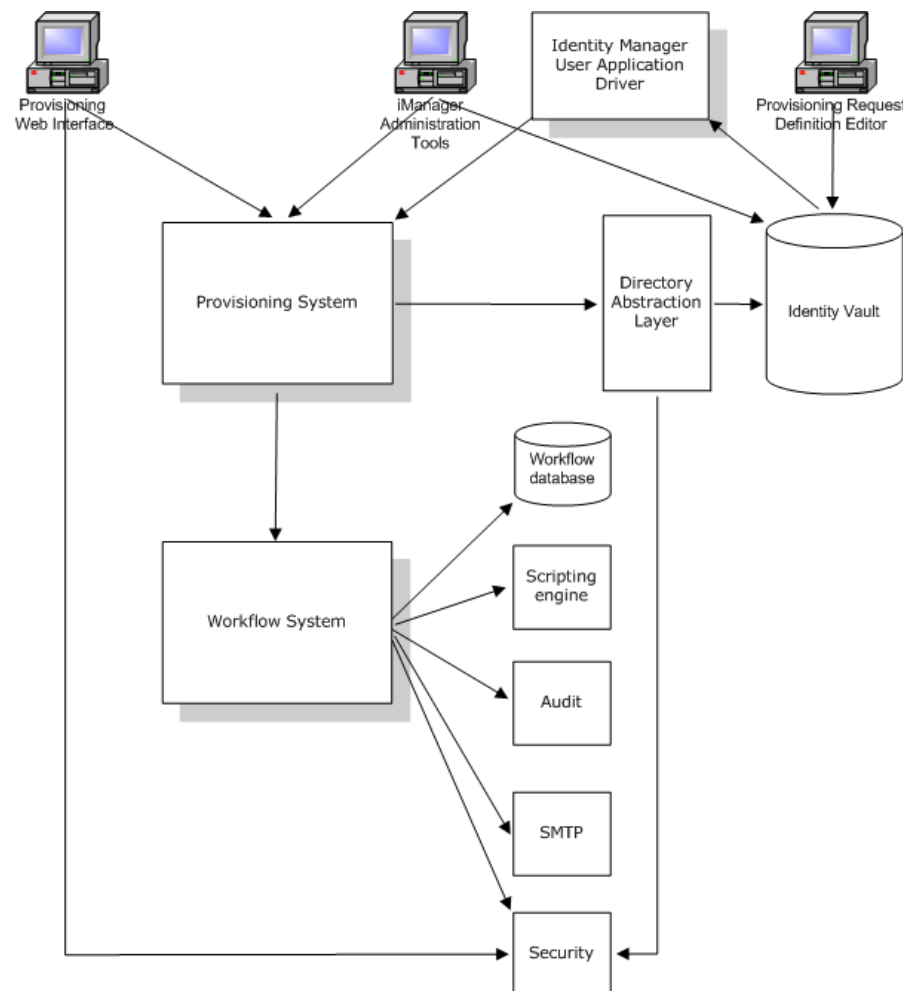
When a provisioning request requires permission from one or more individuals in an organization, the request starts a workflow. The workflow coordinates the *approvals* needed to fulfill the request. Some provisioning requests require approval from a single individual; others require approval from several individuals. In some instances, a request can be fulfilled without any approvals.

Some workflows require that processing proceed in a *sequential* fashion, with each approval step being performed sequentially. Other workflows provide support for *parallel* processing. When you define a provisioning request, you specify whether you want the workflow to support sequential or parallel processing.

To configure a provisioning request, you create a *provisioning request definition*, which binds a resource to a workflow. Identity Manager provides the provisioning request definition editor to give you complete control over the behavior of a provisioning request and its associated workflow. Identity Manager also includes a set of iManager plug-ins that you can use to customize provisioning request definitions that have already been deployed. The iManager tools let you make minor changes to the behavior of a provisioning request definition and also manage workflows that are in process.

The following figure shows how the provisioning request definition editor fits into the workflow-based provisioning system included with Identity Manager:

Figure 4-1 Provisioning Request Definition Editor and the Workflow Architecture



4.1.2 Provisioning and Workflow Example

Suppose a user needs an account on an IT system. To set up the account, the user initiates a request through the Identity Manager User Application. This request starts a workflow, which coordinates

an approval process. When the necessary approvals have been granted, the request is fulfilled. The process includes four basic steps:

- ♦ “Step 1: Initiating the Request” on page 79
- ♦ “Step 2: Approving the Request” on page 79
- ♦ “Step 3: Fulfilling the Request” on page 83
- ♦ “Step 4: Completing the workflow” on page 83

Step 1: Initiating the Request

In the Identity Manager User Application, the user browses a list of resources by *category* and selects one to provision. In the Identity Vault, the *provisioned resource* selected is associated with a provisioning request definition. The provisioning request definition is the most prominent object in a provisioning system. It binds a provisioned resource to a workflow and acts as the means by which the workflow process is exposed to the end user. The provisioning request definition provides all the information required to display the initial request form to the user and to start the flow that follows the initial request.

In this example, the user selects the New Account resource. When the user initiates the request, the Web application retrieves the initial request form and the description of the associated *initial request data* from the Provisioning System, which gets these objects from the provisioning request definition.

When a provisioning request is initiated, the Provisioning System tracks the initiator and the recipient. The *initiator* is the person who made the request. The *recipient* is the person for whom the request was made. In some situations, the initiator and the recipient can be the same individual.

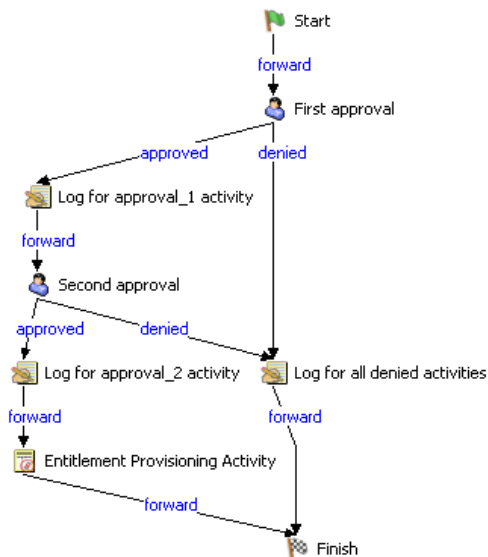
Each provisioning request has an *operation* associated with it. The operation specifies whether the user wants to *grant* or *revoke* the resource.

Step 2: Approving the Request

After the user has initiated the request, the Provisioning System starts the workflow process. The *workflow process* coordinates the approvals. In this example, two levels of approvals are required, one from the user’s manager and a second from the manager’s supervisor. If approval is denied by any user in a workflow, the flow terminates and the request is denied.

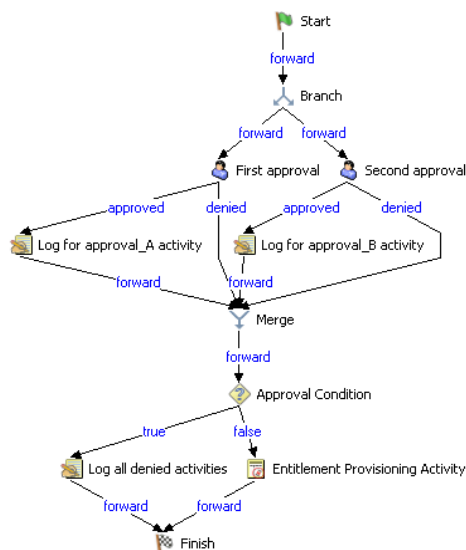
Workflows can process approvals sequentially or in parallel. In a *sequential workflow*, as shown in the following figure, each approval task must be processed before the next approval task begins.

Figure 4-2 *Sequential Workflow with Two Approvals*



In a *parallel workflow*, as shown in the following figure, users can work on the approval tasks simultaneously.

Figure 4-3 *Parallel Workflow with Two Approvals*



NOTE: The display labels (First approval, Second approval, and so on) can easily be changed to suit your application requirements. For parallel flows, you might want to specify labels that do not imply sequential processing. For example, you might want to assign labels such as One of Three Parallel Approvals, Two of Three Parallel Approvals, and so on.

The workflow definition is made up of the components shown in the following table:

Table 4-1 *Workflow Definition Components*

Process Components	Description
Activities	<p>An activity is an object that represents a task. An activity can present information to the user and respond to user interactions. It can also perform background functions that are not visible to the user.</p> <p>In a workflow diagram, the activities are represented by boxes.</p> <p>In the Identity Manager User Application, the activities that handle the approval process are referred to as tasks. An end user can see the list of tasks in his or her queue by clicking <i>My Tasks</i> in the <i>My Work</i> group of actions. To see which workflow activities have been processed for a particular task, the user can select the task and click the <i>View Comment History</i> button on the Task Detail form.</p> <p>To see which workflow activities have been processed for a particular provisioning request, the user can click <i>My Requests</i>, select the request, and click the <i>View Comment and Flow History</i> button on the Request Detail form.</p> <p>For more information on the <i>My Tasks</i> and <i>My Requests</i> actions, see the Identity Manager User Application: User Guide (http://www.novell.com/documentation/idm/index.html).</p>
Flow paths	<p>Flow paths tie the activities in a workflow together. A flow path represents a path to be followed between two activities.</p> <p>An activity can have multiple incoming flow paths and multiple outgoing flow paths. When an activity has more than one outgoing flow path, the flow path selected often depends on the outcome of the activity. The outcome is the end result of processing performed by the activity. For example, an approval activity can have an outcome of approved or denied, depending on the action taken by the user.</p> <p>In a workflow diagram, the flow paths are represented by arrows.</p>

Start activity: The workflow process begins with the execution of the Start activity. This activity displays the initial request form to the user. Once the user has provided the initial request data, it initializes a work document using this data. The Start activity also binds several system values, such as the initiator and recipient, so that these can be used in script expressions.

Approval activities: After the Start activity finishes, the Workflow System forwards processing to the first Approval activity in the flow. The Approval activity sends an e-mail to the approver, notifying this user that their attention is needed. When the user claims the task, the Approval activity displays an approval form, which gives the user the ability to act on the request. In the workflow examples shown in “[Step 2: Approving the Request](#)” on page 79, “First approval” and “Second approval” are examples of Approval activities. The display labels for Approval activities can be localized to satisfy international requirements.

An Approval activity has five possible outcomes, each represented by a different flow path exiting the activity:

- ♦ Approved

- ♦ Denied
- ♦ Refused
- ♦ Error
- ♦ Timeout

NOTE: The Error and Timeout outcomes can occur without any action being taken by the user.

If the user approves the request, the workflow follows the approved flow path to the next activity in the flow. If no further approvals are needed, the resource can be provisioned. If the user denies the request, the workflow follows the denied flow path to the next activity in the flow. Alternatively, the user can reassign the task (if he or she is an Organizational Manager or User Application Administrator), which puts the task in another user's queue.

The user to whom an Approval activity has been assigned is referred to as the addressee. The addressee for an activity can be notified of the assigned task via e-mail. To perform the work associated with the activity, the addressee can click the URL in the e-mail, find the task in the work list (queue), and claim the task.

The addressee must respond to an Approval activity within a specified amount of time; otherwise, the activity times out. Typically the timeout interval is expressed in hours or days to allow the user sufficient time to respond.

When an activity times out, the workflow process might try to complete the activity again, depending on the retry count specified for the activity. In some situations, the workflow process might be configured to escalate an activity that has timed out to another user. In this case, the activity is reassigned to a new addressee (the user's manager, for example) to give this user an opportunity to finish the work of the activity. If the last retry times out, the activity might be marked as approved or denied, depending on how the workflow was configured.

Log activity: The Log activity is a system activity that writes messages to a log. To log information about the state of a workflow process, the Workflow System interacts with Novell® Audit. During the course of its processing, a workflow might log information about various events that have occurred. Users can use the Novell Audit reporting tools to look at logging data.

Condition activity: During the course of execution, a workflow process might perform a test and check the outcome to see what to do next. The Condition activity provides this capability. Condition activities use a scripting expression to define the condition to evaluate. In the workflow examples shown in [“Step 2: Approving the Request” on page 79](#), “Approval Condition” is an example of a Condition activity.

The Condition activity supports three possible outcomes or exit paths:

- ♦ True
- ♦ False
- ♦ Error

Branch and Merge activities In a workflow that supports parallel processing, the Branch activity allows two users to act on different areas of the work item in parallel. After the users have completed their work, the Merge activity synchronizes the incoming branches in the flow.

Step 3: Fulfilling the Request

When a provisioning request has been approved, the Workflow System can begin the provisioning step. At this point, control passes back to the Provisioning System.

To fulfill the provisioning request, the Provisioning System can execute an Identity Manager entitlement or directly manipulate an eDirectory object and its attributes. These actions are performed by either the Entitlement activity or the Entity activity.

Entitlement activity: The Entitlement activity fulfills the provisioning request by granting or revoking an entitlement. This activity is not usually executed unless all of the necessary approvals are given.

Entity activity: The Entity activity fulfills the provisioning request by directly manipulating an eDirectory object and its attributes. This activity is not normally executed unless all of the necessary approvals are given.

Step 4: Completing the workflow

When all other activities have terminated, the workflow executes the Finish activity.

Finish activity: The Finish activity is the final activity in a workflow. When all the activities in a flow have been completed and the final result of the flow is available, the Finish activity is executed. The Finish activity sends a final e-mail notification to inform participants of the completion of the workflow.

4.2 Basic Steps for Creating a Provisioning Request Definition

The following table describes how to define a provisioning request.

Table 4-2 Basic Steps for Defining a Provisioning Request

Task	Action	For More Information
Step 1: Use the wizard to create the provisioning request definition	<p>Provide a name for the provisioning request and define its basic characteristics. Then, specify whether you want to use a template to create the request.</p> <p>Provisioning request definitions are stored locally in the <code>Provisioning\AppConfig\RequestDefs</code> directory within your workspace.</p>	See Chapter 5, "Creating a Provisioning Request Definition," on page 95.

Task	Action	For More Information
Step 2: Create the forms	Create the initial request and approval forms for the workflow. By creating the forms first, you can ensure that the user interface is correct before proceeding to the implementation details. In addition, you can greatly simplify the process of mapping the form fields to the application data.	See Chapter 6, “Creating Forms for a Provisioning Request Definition,” on page 103.
Step 3: Create the workflow diagram	Add the activities to the workflow diagram and connect them with flow paths.	See Chapter 7, “Creating the Workflow for a Provisioning Request Definition,” on page 157
Step 4: Configure the activities and flow paths	Specify the properties, data item mappings, and e-mail notification settings for the activities. Then, define the semantics for the flow paths.	See Chapter 8, “Workflow Activity Reference,” on page 179

4.3 Guidelines for Creating Workflows

To create well-formed workflows, you need to understand the rules for adding activities and flow paths. In addition, you need to understand how to manipulate workflow data. See the following topics:

- ♦ [Section 4.3.1, “Rules for Activities,”](#) on page 84
- ♦ [Section 4.3.2, “Rules for Flow Paths,”](#) on page 85
- ♦ [Section 4.3.3, “Understanding Workflow Data,”](#) on page 86

NOTE: You can validate a provisioning request definition before you deploy it. For more information, see [Section 2.6, “Validating Provisioning Objects,”](#) on page 28.

4.3.1 Rules for Activities

When adding activities to a workflow, follow these rules:

- ♦ A workflow must have only one Start activity and one Finish activity.
- ♦ A workflow can have zero or more of the following activity types:
 - Approval activity
 - Branch activity
 - Condition activity
 - Log activity
 - Mapping activity
 - Merge activity
- ♦ Each Branch activity must have a corresponding Merge activity.
- ♦ To ensure that the provisioning step is performed, a workflow must have at least one Entitlement activity or Entity activity.

4.3.2 Rules for Flow Paths

When adding flow paths to a workflow, follow these rules:

- ♦ With the exception of the Start activity, all activities can have one or more incoming flow paths. The Start activity cannot have any incoming flow paths.
- ♦ The Finish activity cannot have any outgoing flow paths.
- ♦ There can be only one flow path out of the Start activity. The flow path type must be forward.
- ♦ There can be between one and five flow paths out of the Approval activity. The valid flow path types are approved, denied, refused, timedout, and error. At runtime, only one of the flow paths is executed.
- ♦ There can be only one flow path out of the Entitlement, Entity, Log, and Merge activities. The flow path type must be forward.
- ♦ There can be two or three flow paths out of the Condition activity. The valid flow path types are true, false, and error. The true and false flow paths are required; the error flow path is optional.
- ♦ There can be one or more flow paths out of the Branch activity. The flow path type must be forward for each path. At runtime, all of the flow paths execute.

The following table summarizes the rules for adding flow paths into and out of an activity:

Table 4-3 *Number of Flow Paths Permitted for Each Activity*

Activity	Inbound Paths	Outbound Paths
Start	0	1 Must always be forward.
Approval	1 to n	1 to 5 Approved, denied, refused, timedout, or error.
Entitlement	1 to n	1 Must always be forward.
Entity	1 to n	1 Must always be forward.
Log	1 to n	1 Must always be forward.
Condition	1 to n	2 to 3 True and false are required; error is optional.
Branch	1 to n	1 to n
Merge	1 to n	1 Must always be forward.
Finish	1 to n	0

Activity	Inbound Paths	Outbound Paths
Mapping	1 to n	1

The following table summarizes which activity types can be a source or target for each of the available flow path types:

Table 4-4 *Flow Path Types Allowed for Each Activity*

Activity	Forward	Approved	Denied	Refused	Timeout	True	False	Error
Start	Source							
Approval	Target	Source/ Target	Source/ Target	Source/ Target	Source/ Target	Target	Target	Source/ Target
Entitlement	Source/ Target	Target	Target	Target	Target	Target	Target	Target
Entity	Source/ Target	Target	Target	Target	Target	Target	Target	Target
Log	Source/ Target	Target	Target	Target	Target	Target	Target	Target
Condition	Target	Target	Target	Target	Target	Source/ Target	Source/ Target	Source/ Target
Branch	Source/ Target	Target	Target	Target	Target	Target	Target	Target
Merge	Source/ Target	Target	Target	Target	Target	Target	Target	Target
Finish	Target	Target	Target	Target	Target	Target	Target	Target
Mapping	Source	Target	Target	Target	Target	Target	Target	Target

4.3.3 Understanding Workflow Data

When you're creating a workflow, you can manipulate workflow data to suit the needs of your provisioning application.

The workflow uses a single process object to manage information about the process. A separate activity object is created for each activity in the workflow and form data is maintained for each activity that provides for user interaction.

The data objects associated with each user interface control on a form (text field, drop down list, and so forth) can be modified immediately prior to the execution of the corresponding activity (Start activity or Approval activity). In addition, this data can be retrieved immediately after execution of the activity. Once control has been passed to the next activity, the form control data is no longer available. For this reason, the workflow provides a special object called flowdata that allows you to define your own data items. You can add your own variables to this object to keep track of information that is important to your workflow, including form data that would otherwise be lost.

The following table summarizes the categories of workflow data:

Table 4-5 *Categories of Workflow Data*

Data object	Lifetime	Editable	Creator
process	Workflow	No	System
activities	Workflow	No	System
activity forms	Activity	Yes	System and workflow designer
flowdata	Workflow	Yes	Workflow designer

NOTE: The workflow designer is the person who creates the workflow in Designer.

The following table describes the variables for each type of object:

Table 4-6 *Data Variables in a Workflow*

Object	Variable	Description
process	approvalStatus	The current status of the process.
	category	The provisioning category (for example, Entitlements) selected by the person who initiated the request.
	container dn	The distinguished name of the container defined for the user application at install time.
	description	The description of the provisioning request definition.
	group container dn	The distinguished name of the group container defined for the user application at install time.
	id	The unique IDVault id (CN) of the provisioning request definition.
	initiator	The distinguished name of the person who initiated the request.
	locale	The current locale.
	name	The workflow process name.
	provisioning driver dn	The distinguished name of the provisioning driver defined for the user application at install time.
	recipient	The distinguished name of the intended target of the provisioned resource.
	user container dn	The distinguished name of the user container defined for the user application at install time.

Object	Variable	Description
<i>approval-activity-name</i>	requestID	The ID for the provisioning request.
	timestamp	The time the process was initiated.
	action	The action taken by the user.
	addressee	The current addressee for the approval activity.
	name	The name of the activity.
	timestamp	The time that the activity was queued on the work list.
	user	The user who is associated with the current activity.
<i>form-name</i>	workId	The system generated unique id of the current workflow activity.
	<i>custom-form-controls</i>	Any user interface control you add to a form.
flowdata	<i>custom-variables</i>	Any custom variables you create to hold data needed for the workflow. If you use one of the installed templates to create your workflow, the flowdata object can have a variable called <i>reason</i> , which contains text copied from the reason field on the initial request form.

You can reference these objects in ECMAScript expressions. Script expressions in a workflow can at any time refer to data items that are bound upstream in the flow. However, workflow expressions cannot refer to data items that are created downstream (because these data items don't exist yet) or to data bound on other branches in a flow that supports parallel processing (because these branches could be executing concurrently with the current activity).

Creating New Data Items

You can create a new data item on the flowdata object by specifying a post-activity target expression on the *Data Item Mapping* tab for the Start or Approval activities. If you specify a name for a new data item in the *Target Expression* column, this automatically creates the variable. Any activity executed after this activity can then access the data item.

For example, you might want to map the form field called *reason* to the target expression flowdata.myReason. The variable myReason then becomes a new data item that is available to all activities executed later in the workflow.

Modifying Data Items

You can modify a data item by specifying a pre-activity expression on the *Data Item Mapping* tab for the Start or Approval activities. For example, to prepend a dollar sign to a price, you might map the following source expression to a target form field called *Price*:

```
"$" + flowdata.get('cost')
```

When the form displays to the user, the Price data appears as follows:

```
$xx.xx
```

Another example might be computing the total cost by adding the tax to the base cost. To do this, you could map the following source expression to a target form field called TotalCost:

```
Number(flowdata.get('cost')) + Number(flowdata.get('tax'))
```

Working with Complex Data Item Mappings

All data in the flowdata object is maintained in XML, so you can create data items in a hierarchical fashion as well. For example, suppose you have a workflow form that allows a user to ask for access to two internal systems, one for accounts payable and one for receivables. Suppose the form has (among other fields) two Yes/No fields named *Acct_Pay* and *Acct_Rec*. In the post-activity data item mappings, you might create two mappings as follows:

Table 4-7 Complex Data Item Mapping Examples

Source Form Field	Target Expression
<i>Acct_Pay</i>	flowdata.SystemAccess/AcctPay
<i>Acct_Rec</i>	flowdata.SystemAccess/AcctRec

This would create an XML element named *SystemAccess* with two child elements named *AcctPay* and *AcctRec*. One reason to structure data in this way is for clearer organization and management of data in complex work flows containing many forms and data items. To retrieve data from these hierarchies, the following syntax would be used:

```
flowdata.get('SystemAccess/AcctPay') .
```

For complete details on building ECMAScript expressions, see [Chapter 10, “Working with ECMA Expressions,”](#) on page 249.

Moving Form Control Data to Flowdata

All form controls you create (except for *DNDisplay*) are automatically made available for use in pre-activity and post-activity expressions on the *Data Item Mapping* tab for the activity that uses the form. For example, suppose you want to make a user's entry data in control *ACONTROL* on form *AFORM* in *AACTIVITY* available for use in a subsequent activity. To do this, you would select *AACTIVITY* in the workflow, select the *Data Item Mapping* tab, and click the *Post Activity Mapping* radio button. Next to the source form field *ACONTROL*, you would then enter a target expression in the following format:

```
flowdata.my_ACONTROL
```

Any subsequent activity in the workflow would then be able to access this data by using pre-activity source expressions such as these:

```
flowdata.get('my_ACONTROL')
flowdata.getObject('my_ACONTROL')
```

Moving Flowdata to Form Controls

You can also move flowdata values into form controls. The simplest case is moving a single text value into a form control. In the example above, suppose ACONTROL is a simple text entry field. In this case, to move it into another text entry field in an activity called ZACTIVITY, you would select ZACTIVITY in the workflow, select the *Data Item Mapping* tab, and click the *Pre Activity Mapping* radio button. Next to the target form field, you would then enter this source expression:

```
flowdata.my_ACONTROL
```

To move more complex form control data (for example, a MultiValue DN control) into another form control, you can use the getObject() expression syntax. For example, assuming ACONTROL is a MultiValue DN control, you could use this source expression:

```
flowdata.getObject('my_ACONTROL')
```

To move data into a form control, you need to be aware of type constraints. For example, you should not try to move text-based data into a numeric control, or a boolean value into a DN control.

4.4 Working with the Installed Templates

Identity Manager ships with a set of preconfigured provisioning request definitions and workflows. You can use these as templates for building your own provisioning system. To set up your system, you define new objects based on the installed templates and customize these objects to suit the needs of your organization.

The installed templates let you determine the number of approval steps required for the request to be fulfilled. You can configure a provisioning request to require from zero to five approval steps:

You can also specify whether you want to support sequential or parallel processing, and whether you want to approve or deny the request in the event that the workflow times out during the course of processing.

The following table lists the templates included with Identity Manager.

Table 4-8 *Preconfigured Provisioning Request Definitions and Workflows*

Template	Description
Self Provision Approval	Allows a provisioning request to be fulfilled without any approvals.
One Step Approval (Timeout Approves)	Requires a single approval for the provisioning request to be fulfilled. If an activity times out, the activity approves the request and the work item is forwarded to the next activity.
Two Step Sequential Approval (Timeout Approves)	Requires two approvals for the provisioning request to be fulfilled. If an activity times out, the activity approves the request and the work item is forwarded to the next activity. This template supports sequential processing.

Template	Description
Three Step Sequential Approval (Timeout Approves)	<p>Requires three approvals for the provisioning request to be fulfilled. If an activity times out, the activity approves the request and the work item is forwarded to the next activity.</p> <p>This template supports sequential processing.</p>
Four Step Sequential Approval (Timeout Approves)	<p>Requires four approvals for the provisioning request to be fulfilled. If an activity times out, the activity approves the request and the work item is forwarded to the next activity.</p> <p>This template supports sequential processing.</p>
Five Step Sequential Approval (Timeout Approves)	<p>Requires five approvals for the provisioning request to be fulfilled. If an activity times out, the activity approves the request and the work item is forwarded to the next activity.</p> <p>This template supports sequential processing.</p>
One Step Approval (Timeout Denies)	<p>Requires a single approval for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</p>
Two Step Sequential Approval (Timeout Denies)	<p>Requires two approvals for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</p> <p>This template supports sequential processing.</p>
Three Step Sequential Approval (Timeout Denies)	<p>Requires three approvals for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</p> <p>This template supports sequential processing.</p>
Four Step Sequential Approval (Timeout Denies)	<p>Requires four approvals for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</p> <p>This template supports sequential processing.</p>
Five Step Sequential Approval (Timeout Denies)	<p>Requires five approvals for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</p> <p>This template supports sequential processing.</p>
Two Step Parallel Approval (Timeout Approves)	<p>Requires two approvals for the provisioning request to be fulfilled. If an activity times out, the activity approves the request and the work item is forwarded to the next activity.</p> <p>This template supports parallel processing.</p>

Template	Description
Three Step Parallel Approval (Timeout Approves)	<p>Requires three approvals for the provisioning request to be fulfilled. If an activity times out, the activity approves the request and the work item is forwarded to the next activity.</p> <p>This template supports parallel processing.</p>
Four Step Parallel Approval (Timeout Approves)	<p>Requires four approvals for the provisioning request to be fulfilled. If an activity times out, the activity approves the request and the work item is forwarded to the next activity.</p> <p>This template supports parallel processing.</p>
Five Step Parallel Approval (Timeout Approves)	<p>Requires five approvals for the provisioning request to be fulfilled. If an activity times out, the activity approves the request and the work item is forwarded to the next activity.</p> <p>This template supports parallel processing.</p>
Two Step Parallel Approval (Timeout Denies)	<p>Requires two approvals for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</p> <p>This template supports parallel processing.</p>
Three Step Parallel Approval (Timeout Denies)	<p>Requires three approvals for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</p> <p>This template supports parallel processing.</p>
Four Step Parallel Approval (Timeout Denies)	<p>Requires four approvals for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</p> <p>This template supports parallel processing.</p>
Five Step Parallel Approval (Timeout Denies)	<p>Requires five approvals for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</p> <p>This template supports parallel processing.</p>

4.5 Debugging a Workflow

When you're testing a workflow, you might need to see the values of the variables you're using in the flow. There are several ways to do this. One approach is to use the Log activity to display messages containing the variables you need to look at. After you've configured the Log activity, you can then see the messages in the console. In the Log activity, you can use scripting expressions in the Message property to retrieve the values you need. For example, you might use this expression to log a message containing the value of a variable defined on the flowdata object:

```
flowdata.get('my_variable')
```

For details on using the Log activity, see [Section 8.3, “Log Activity,” on page 188](#).

Another approach is to look in the workflow database to see how the data associated with the flowdata object changes as the workflow progresses from one activity to the next. To see this data, you can look at the `afdocument` table.

A final approach you can use during the debugging process is to change the log levels associated with the workflow system (`com.novell.soa.af.impl`), the provisioning requests component of the User Application (`com.novell.srpr.apwa`), and the evaluation of server side scripts (`com.novell.soa.script`). This approach might generate more information than you need, but sometimes it can be helpful. To change logging levels, go to the Logging page within the Administration tab of the User Application. For more information about logging, see “**Setting Up Logging**” in the *Identity Manager 3.5 User Application: Administration Guide*.

Creating a Provisioning Request Definition

5

This section provides details about creating a provisioning request definition. Topics include:

- ♦ [Section 5.1, “About the Wizard and the Overview Tab,” on page 95](#)
- ♦ [Section 5.2, “Using the Wizard to Create a Provisioning Request Definition,” on page 97](#)
- ♦ [Section 5.3, “Using the Overview Tab to Modify Basic Settings,” on page 100](#)

5.1 About the Wizard and the Overview Tab

You create provisioning request definitions in three main steps:

- ♦ Create the basic information about the provisioning request definition (for example, the name of the provisioning request definition, the category to which it belongs, who can access it) using the Create A New PRD Wizard. After you have created the basic provisioning request definition, the basic information is displayed in the *Overview* tab.
- ♦ Create the forms that interact with the workflow participants using the *Forms* tab.
- ♦ Design the workflow using the *Workflow* tab.

To add a provisioning request definition:

1 Launch the Create A New PRD Wizard in one of these ways:

- ♦ From the Provisioning view, right-click the *Provisioning Request Definitions* node and choose *New*.
- ♦ From the Provisioning view, click a User Application or provisioning request container, then select *Insert > Provisioning Request Definition*.
- ♦ Select *File > New > Provisioning*.
- ♦ Choose *Provisioning Request Definition*, then click *Next*.

The first page of the *Create A New PRD Wizard* is displayed.

2 Fill in the fields as follows:

Field	Description
<i>Identifier (CN)</i>	The CN (common name) identifier for the provisioning request definition.
<i>Display Name</i>	The display name for the provisioning request definition. This is the name that is displayed in the provisioning view.
<i>Description</i>	A description of the provisioning request definition.

3 Click *Next*. The next page of the wizard is displayed.

You can create new provisioning request definitions based on a template, or you can build the provisioning request definition from concept to finished product. Use the next panel of the wizard to specify whether or not to base this provisioning request definition on a template.

4 Perform one of these steps:

- ♦ To base this provisioning request definition on a template, select *Create a provisioning request definition using one of the templates*, then select the desired template (for example, TemplateSingleApproval_TA) from the *Available Templates* list, then click *Next*.
- ♦ To build this provisioning request definition from concept to finished product, click *Next*.

You use the next panel of the wizard to specify the trustees (in other words, users) who can access the provisioning request definition after it is deployed.

The screenshot shows a Windows-style dialog box titled "Create a new PRD". The main heading inside is "Specify the details of the Provisioning Request Definition." Below this, there is a "Category:" label followed by a dropdown menu currently showing "Accounts". Underneath the dropdown is a checked checkbox labeled "Notify participants by email". To the left of a table is the label "Trustee rights:" followed by two small icons: a green plus sign and a yellow minus sign. The table itself has a header row labeled "Trustee DN" and several empty rows below it. At the bottom of the dialog, there are four buttons: a help button (question mark icon), "< Back", "Next >", and "Finish". A "Cancel" button is also present to the right of the "Finish" button.

5 Click the plus (+) icon to add a trustee.

Designer displays a panel that allows you to browse the Identity Vault to select a trustee. You can select an individual trustee or a group.

If you cannot connect to the Identity Vault, you can type trustee DNs directly in the *Trustee DN* field.

6 Select the trustee, then click *OK*.

Designer returns you to the previous panel. If desired, add additional trustees by repeating the previous step.

7 When you have finished adding trustees, click *Finish*.

Designer displays the Provisioning Request Definition Details panel on the *Overview* tab (see [Section 5.3, "Using the Overview Tab to Modify Basic Settings," on page 100](#)).

5.2 Using the Wizard to Create a Provisioning Request Definition

You can create a provisioning request definition with a template or from concept to finished product. We recommend that you use an existing template to create new provisioning request definitions. This saves time and allows you to make targeted changes to an existing provisioning request

definition. However, if no existing provisioning request definition resembles new work that you want to do, you can create a new provisioning request from concept to finished product.

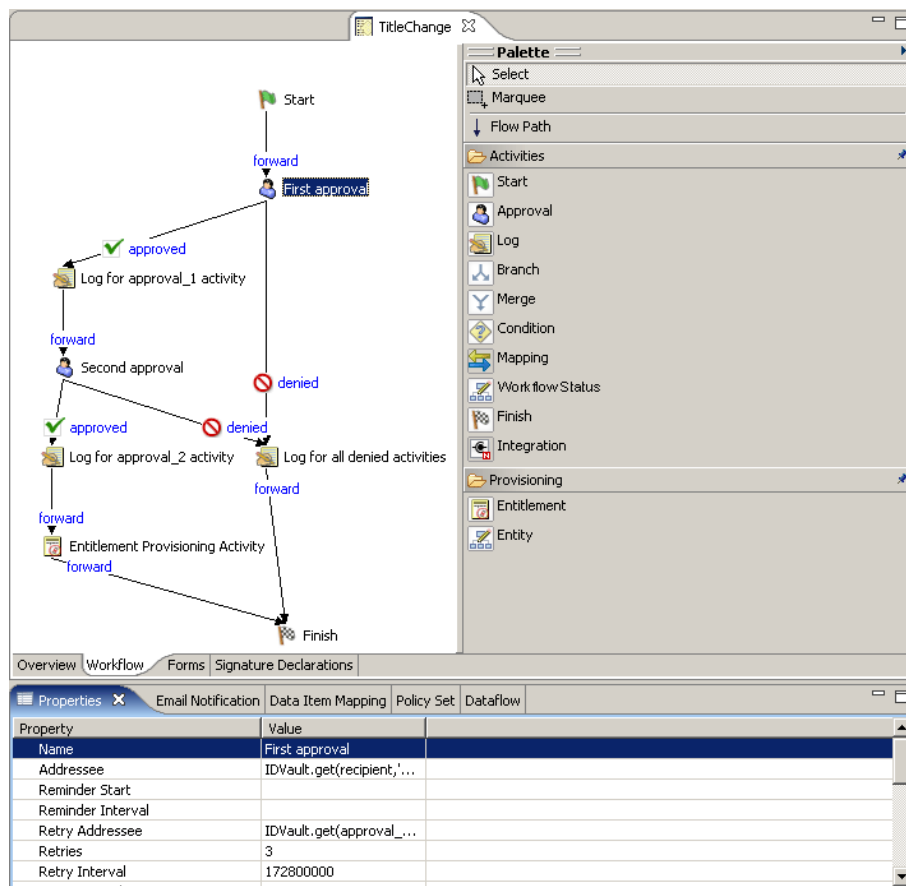
See the following topics:

- ♦ [Section 5.2.1, “Using a Template,” on page 98](#)
- ♦ [Section 5.2.2, “From Concept to Finished Product,” on page 99](#)

5.2.1 Using a Template

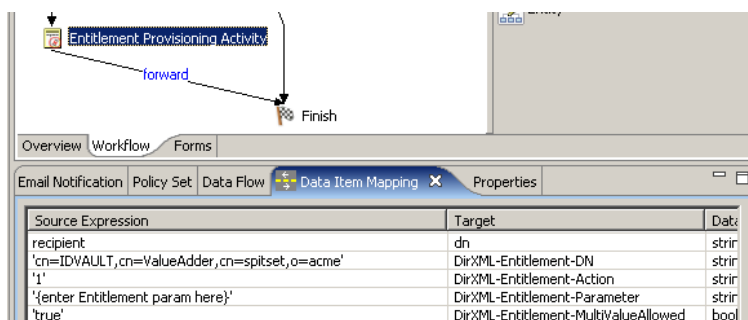
To create a provisioning request definition using a template:

- 1 Create the basic information for a new provisioning request definition (see [Section 5.1, “About the Wizard and the Overview Tab,” on page 95](#)). In step [Step 4 on page 97](#), select *Create a provisioning request definition using one of the templates*, then select the desired template. When you are finished, the *Overview* tab for the new provisioning request is displayed.
- 2 Click the *Workflow* tab. The Workflow view is displayed.



The provisioning request definition template includes some default values that you will want to customize for your environment. For example, the Entitlement Provisioning Activity contains placeholder values for several data item mapping properties. You need to replace the placeholder values with the actual values for your provisioning request.

- 3 Click the Entitlement Provisioning activity, then click the *Data Item Mapping* tab.



- 4 Double-click in the *Source Expression* field for the *DirXML-Entitlement-DN* target field, then click the button that appears in the field to display the ECMA expression builder.
- See [Chapter 10, “Working with ECMA Expressions,” on page 249](#) for information about the ECMA expression builder.
- 5 Use the ECMA expression builder to replace the placeholder expression with an expression that specifies the entitlement that you would like to provision with this provisioning request.
 - 6 Replace the placeholder expression in the *Source Expression* field for the *DirXML-Entitlement-Parameter*.
 - 7 Click the *Forms* tab and customize the forms for the provisioning request to your needs.
- See [Chapter 6, “Creating Forms for a Provisioning Request Definition,” on page 103](#). The template includes predefined request and approval forms. You might want to add additional forms, or add or remove form controls.
- 8 Click the *Workflow* tab and customize the properties of the workflow to your needs.
- See [Chapter 7, “Creating the Workflow for a Provisioning Request Definition,” on page 157](#) and [Chapter 8, “Workflow Activity Reference,” on page 179](#).

5.2.2 From Concept to Finished Product

Whenever possible, use an existing template (or save an existing provisioning request definition under a new name) to create new provisioning request definitions. This saves you time and allows you to make targeted changes to an existing provisioning request definition. However, if no existing provisioning request definition resembles the new work that you want to do, then you need to build a provisioning request definition from concept to finished product. You can still save time and effort by re-using forms from other workflows.

To create a provisioning request definition:

- 1 Create the basic information for a new provisioning request definition (see [Section 5.1, “About the Wizard and the Overview Tab,” on page 95](#)). In step [Step 4 on page 97](#), do not select *Create a provisioning request definition using one of the templates*, and do not select a template. When you are finished, the *Overview* tab for the new provisioning request is displayed.
- 2 Create the forms for the provisioning request definition. After you have created the basic provisioning request definition, the next step is to create the forms that are presented to the provisioning request users. It's important to define forms *before* you create the workflow topology. This ensures that data bindings can be set up automatically for each activity when you create activities.

There are two types of forms:

request: Used by the person requesting the resource to specify the item or capability that is being requested. One request form can be defined for a workflow. The request form is always associated with the Start Activity.

approval: Used by the person receiving the provisioning request to approve, refuse, or comment on the provisioning request. One or more approval forms can be defined. You must associate each form with an Approval activity. You associate an approval form with an Approval activity by using the properties for the activity.

To create the forms, see [Section 6.3, “Creating forms,” on page 111](#).

- 3 Click the *Workflow* tab and create the workflow topology.

You create the topology of a workflow by creating and linking activities into the desired workflow pattern, and by assigning rules to the flowpaths between activities. For information about creating a workflow topology, see [Chapter 7, “Creating the Workflow for a Provisioning Request Definition,” on page 157](#).

- 4 Specify the details (properties, data item mappings, e-mail notification) for each workflow activity.

To specify workflow activity details, see [Chapter 8, “Workflow Activity Reference,” on page 179](#).

- 5 Configure the flowpaths between workflow activities.

To configure flowpaths, see [Section 7.4, “Configuring Flow Paths,” on page 165](#).

5.3 Using the Overview Tab to Modify Basic Settings

You use the *Overview* tab to define the basic information about the provisioning request definition (for example, the name of the provisioning request definition, the category to which it belongs, who can access it).

Figure 5-1 Overview Tab

The screenshot shows a window titled "TitleChange" with a tab labeled "TitleChange". Inside the window is a form titled "Provisioning Request Definition Details". The form contains the following fields and options:

- Identifier (CN): TitleChange
- Display Name: TitleChange
- Description: TitleChange
- Category: Entitlements (dropdown menu)
- Status: Inactive (dropdown menu)
- Flow Strategy: Single Flow (dropdown menu)
- ☒ Notify participants by email
- ☐ Restrict View
- ☒ Generate Comments
- ☐ Set Default Completion Status to Approved
- Trustee rights: + X
- Trustee DN: HR.groups.idm-leda.novell, IT.groups.idm-leda.novell

At the bottom of the window are four tabs: Overview, Workflow, Forms, and Signature Declarations. The "Overview" tab is currently selected.

The following table describes each property that you can configure on the *Overview* tab.

Table 5-1 Overview Properties

Property	Description
<i>Identifier (CN)</i>	Displays the CN (common name) of the provisioning request definition. The CN cannot be changed.
<i>Display Name</i>	Specifies the display name of the provisioning request definition. This is the name that is displayed to the user in Designer and Identity Manager.
<i>Description</i>	Specifies the description of the provisioning request definition.
<i>Category</i>	Specifies the category to which the provisioning request definition belongs from the list of Provisioning Categories defined in the directory abstraction layer.
<i>Status</i>	<p>Specifies the status of the provisioning request definition:</p> <p>Active: Select to make the provisioning request definition available for use in the User Application.</p> <p>Inactive: Select to make the provisioning request definition temporarily unavailable for use in the User Application. You can use this option when you want keep the roles of the person who develops and deploys the provisioning request definition separate from the person who activates the provisioning request definition. For example, a developer could deploy the provisioning request definition as Inactive, and an administrator could be responsible for changing the status to Active.</p> <p>Template: Select if you want to use this provisioning request definition as the basis for other provisioning request definitions. Templates are not available for use in the User Application.</p> <p>Retired: Select to mark the provisioning request definition as permanently unavailable for use in the User Application (you can still change the status of the provisioning request definition at any time). This status provides a way of keeping a historical record of a provisioning request definition that is no longer in use.</p>
<i>Flow Strategy</i>	<p>Specifies the flow strategy for the provisioning request definition:</p> <p>Single Flow: This strategy allows one workflow with one recipient.</p> <p>Flow per member: This strategy allows a single workflow to spawn multiple workflows for each member of a group, in which each member is approved individually.</p> <p>Single Flow Provision Members: This strategy allows a single workflow to approve all group members and spawn multiple provisioning steps (one for each group member).</p>
<i>Notify participants by email</i>	<p>Specifies whether approvers are notified by e-mail about pending approval tasks, and whether initiators are notified by e-mail of workflow completion. If Notify participants by email is not checked, then users must look at the <i>Requests and Approvals</i> tab in the User Application for notifications about tasks.</p> <p>For information about selecting an e-mail template and customizing e-mail template parameters, see Section 8.9, "Finish Activity," on page 193.</p>
<i>Restrict View</i>	Specifies that the list of tasks that can be viewed by the user in My Requests in the User Application is restricted to tasks initiated by the user. The default behavior (that is, Restrict View is not checked) is that the user can see any requests that the user initiated or for which the user is the recipient.

Property	Description
<i>Generate Comments</i>	Specifies that the workflow engine should generate comments as the workflow progresses. These comments can be displayed by clicking the following: <ul style="list-style-type: none"> ♦ <i>View Comment and Flow History</i> in a <i>Request Detail</i> form in the User Application ♦ <i>View Comment History</i> on a <i>Task Detail</i> form in the User Application
<i>Set Default Completion Status to Approved</i>	Specifies that the default completion status of the provisioning request is approved if checked, or denied if not checked. This feature can be useful for provisioning requests that do not contain a provisioning activity (an Entitlement or Entity). The value of this parameter can be overridden by explicitly setting the completion status using a provisioning activity or Workflow Status activity.
<i>Trustee Rights</i>	Specifies the users and groups that can use the provisioning request definition.

Creating Forms for a Provisioning Request Definition

6

This section provides details on creating and customizing the User Application's request and approval forms. Topics include:

- ♦ [Section 6.1, “About Forms,” on page 103](#)
- ♦ [Section 6.2, “About the Forms Tab,” on page 107](#)
- ♦ [Section 6.3, “Creating forms,” on page 111](#)
- ♦ [Section 6.4, “Action Reference,” on page 120](#)
- ♦ [Section 6.5, “Form Control Reference,” on page 122](#)
- ♦ [Section 6.6, “Working with Distinguished Names,” on page 147](#)
- ♦ [Section 6.7, “Using DAL Queries in Forms,” on page 151](#)

6.1 About Forms

Forms allow the user to request a resource, approve a resource request, and work on a task. They are available when the user chooses any of the actions in the *My Work* category of the User Application's *Requests and Approvals* tab. [Figure 6-1](#) is an example of a resource request form. At the top of the form is a read-only area that displays the details of the request (or approval for approval forms). In the *Form Detail* section at the bottom, the user provides input to the resource request (or approval) and takes some action on it.

Figure 6-1 Sample Form

The screenshot shows the Novell Identity Manager interface. The top navigation bar includes 'Novell Identity Manager', 'Welcome, Allison', 'Identity Self-Service', 'Requests & Approvals', 'Logout', and 'Help'. The left sidebar lists 'My Work', 'My Tasks', 'Request Resource', 'My Requests', 'My Settings', 'Enter Proxy Mode', 'Edit Availability', 'My Proxy Assignments', and 'My Delegate Assignments'. The main content area is titled 'Request Resource' and displays 'Step 3 of 3: Confirm and complete resource request.' It includes a read-only section with the following details: Resource: Title Change Request, Recipient: Allison Blake, Resource Category: Accounts, and Description: Allow a user to request a title change. Requires manager approval. Below this is the 'Form Detail' section, which contains a 'One Step Approval (Timeout Approves)' form. This form has fields for Recipient (Allison Blake), Current Title (NewTitle), Requested New Title, and Reason for request (marked with an asterisk). There are 'Submit' and 'Cancel' buttons at the bottom of the form.

You use the *Forms* tab of the provisioning request definition editor to define the appearance and behavior of the *Form Detail* section of the User Application's requests and approvals forms.

About Request Forms

You can create one request form for a provisioning request definition. The request form is associated with the workflow's **Start Activity**.

Figure 6-2 Sample Resource Request Form

The screenshot shows the 'Request Resource' form in the Novell Identity Manager interface. The form is titled 'Request Resource' and is part of a 'Step 3 of 3: Confirm and complete resource request.' process. It includes a sidebar with navigation links like 'My Work', 'My Tasks', 'Request Resource', 'My Requests', 'My Settings', 'Enter Proxy Mode', 'Edit Availability', 'My Proxy Assignments', and 'My Delegate Assignments'. The main content area displays the following information:

- Resource:** Title Change Request
- Recipient:** Allison Blake
- Resource Category:** Accounts
- Description:** Allow a user to request a title change. Requires manager approval.

Below this information is a 'Form Detail' section with the heading 'One Step Approval (Timeout Approves)' and the instruction 'Press 'Submit' to request the entitlement.' The form includes fields for 'Recipient' (Allison Blake), 'Current Title' (NewTitle), 'Requested New Title' (empty), and 'Reason for request' (empty). There are 'Submit' and 'Cancel' buttons at the bottom.

About Approval Forms

You can define multiple approval forms for a provisioning request definition, but only one form per **Approval Activity**. You link an approval form to an approval activity in the properties for the activity.

Figure 6-3 Sample Resource Approval Form

The screenshot shows the 'Single Approval' form in the Novell Identity Manager interface. The form is titled 'Single Approval' and is part of a 'Please select the appropriate button to approve or reject the request.' process. It includes a sidebar with navigation links like 'My Requests', 'My Settings', 'Enter Proxy Mode', 'Edit Availability', 'My Proxy Assignments', 'My Delegate Assignments', 'My Team's Work', 'Team Tasks', 'Request Team Resources', 'Team Requests', 'My Team's Settings', 'Team Proxy Assignments', 'Team Delegate Assignments', and 'Team Availability'. The main content area displays the following information:

- Resource:** Title Change Request
- Recipient:** Allison Blake
- Requested By:** Allison Blake
- Task:** Single Approval
- In Queue since:** 06/30/2006 12:32:18 PM
- Timeout on:** 03/17/2026 12:32:18 PM
- Assigned To:** Margo MacKenzie
- Claimed By:**

Below this information are buttons for 'Claim', 'Release', 'Reassign', and 'Back'. The 'Form Detail' section includes the heading 'Single Approval' and the instruction 'Please select the appropriate button to approve or reject the request.' The form includes fields for 'Requested by' (Allison Blake), 'Request Date' (06/30/2006), 'Reason' (Test), 'Current Employee Title' (NewTitle), and 'Requested Title Change' (Marketing Assistant). There is a 'Comment' field and a 'View Comment History' button at the bottom.

6.1.1 About Form Control Data Binding

All of the fields you define for a form are automatically available for data binding in the Data Item Mapping property sheet. Two bindings, or mappings, are possible for each form field: a pre-activity mapping to initialize or pre-load a form field with data, and a post-activity mapping to move modified form data into the work flow document called flowdata. These data-item bindings and any script expressions they utilize execute on the application server as preparation of the form before it is sent to the client browser for display to the user. Common uses for pre-activity data-item mappings and their expressions that operate against the flow-data document are for moving previous approval data into the current approval or for setting default values for fields. For more information on data item mappings, see [Section 7.2.2, “Defining the Data Item Mappings,” on page 162](#).

Some form controls allow you to initialize their values from data sources other than workflow data. For example, some list controls allow you to specify the initial value as a property of the control. For more information about defining initial values, see [Section 6.5, “Form Control Reference,” on page 122](#).

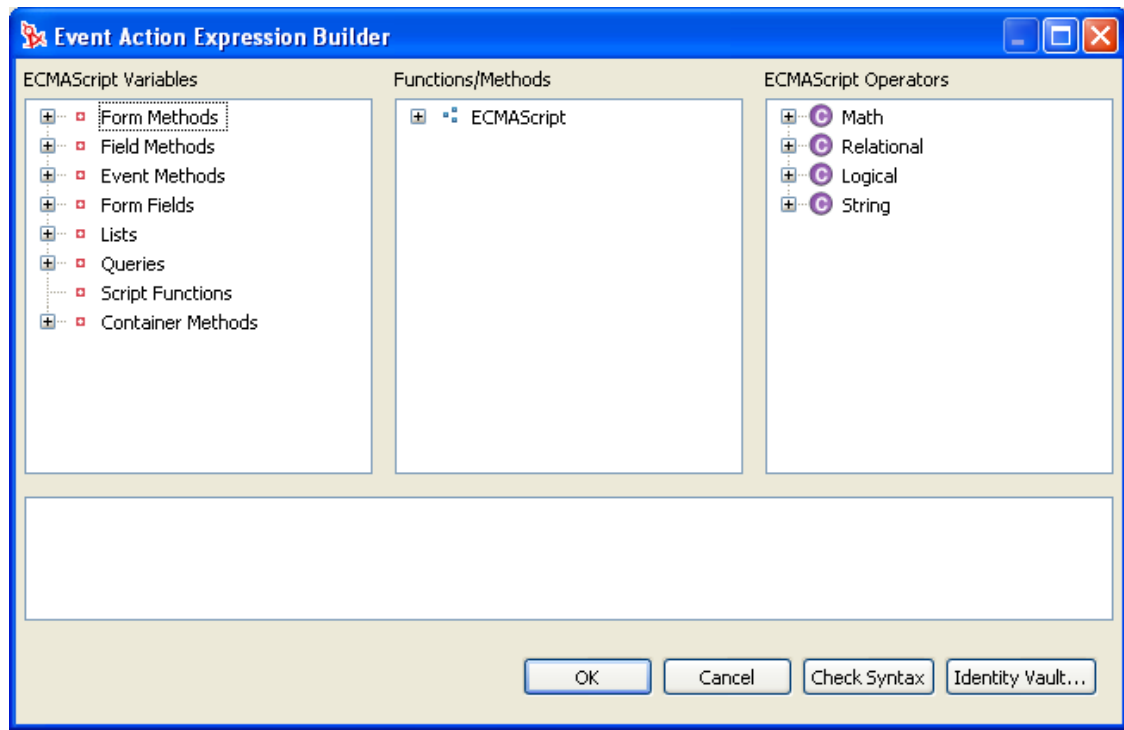
6.1.2 About Forms and Events

Designer allows you to define action scripts that execute on the form control's onLoad, onChange, or custom events. Each form control supports an Events property where you supply the script for the event. The scripts you define have an event-level scope and execute in the browser of the user's client machine.

The Events property provides access to Designer's Event Action Expression Builder, which allows you to create script expressions that refer to and modify form and data. Since Form Control Event

Scripts execute in the client browser, they do not have access to the flow-data document. They do have access to directory abstraction layer Queries.

Figure 6-4 *Event Action Expression Builder*



The Event Action Expression Builder also provides access to the Form Action methods (shown in the left column). This column provides access to the form action script API along with directory abstraction layer query objects. The form action script API is written in JavaScript so that you can add conditions, loops, and user-defined functions. For more information about the Form Action API, see [Section 10.3.1, “Form Action Script Methods,” on page 262](#). To import or include a JavaScript library, you use the Scripts tab of the Form Controls area. For more information, see [Section 6.3.4, “Using the Scripts Tab,” on page 118](#).

6.2 About the Forms Tab

You use the *Forms* tab of the provisioning request definition editor to define the appearance and behavior of your request and approval forms.

Figure 6-5 *Forms Tab*

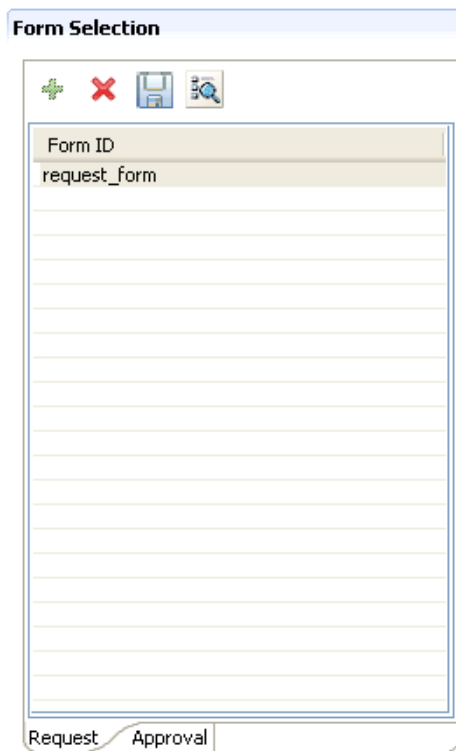
Form Field Name	Data Type	Control Type	Linebreaks
title	string	Title	0
subheading	string	Title	1
recipient	dn	DNDisplay	1
PickList	dn	PickList	1
PickListSingleSelect	dn	PickList	1
Field	dn	MVEEditor	1

The *Forms* tab contains a Form Selection section (described in [About Form Selection](#)) and a Form Controls section (described in [About Form Controls](#)).

6.2.1 About Form Selection





Use the *Form Selection* section to create, delete, or preview a form, or to create a form template. Click the Request or Approval tab depending on the type of form you want to manipulate.

Figure 6-6 *Form Selection*



The *Form Selection* toolbar contains these options:

Table 6-1 *Form Selection Toolbar Options*

Button	Description
	Click to launch the New Form wizard.
	Click to delete an existing form.
	Click to save the form as a template. You can then base other forms on this template. Forms are saved as XML documents in the project directory. Templates are available only within the project in which you create them.
	Click to preview the form.

If you create a provisioning request definition from an existing template, and the template has forms associated with it, the *Form Controls* section displays them. You can modify the form instance using the *Form Controls* section.

6.2.2 About Form Controls

Use the *Form Controls* section to define or modify the form's appearance and behavior.

- ♦ *Fields* tab: Lets you add, delete, and change the data type, control type, and layout order of the controls on the form.

[illegible]

For information about adding controls, see [Section 6.3, “Creating forms,”](#) on page 111. For more information about individual form controls, see [Section 6.5, “Form Control Reference,”](#) on page 122.

- ♦ **Actions** tab: Lets you define the actions the user can perform on the form. Use the *Actions* toolbar to add, delete, and change the actions and layout order of the actions on the form.

[illegible]

For more information about the supported actions, see [Section 6.4, “Action Reference,” on page 120](#).

- ♦ *Scripts* tab: Use the *Scripts* tab to define calls to external JavaScript files or to write JavaScript scripts that are stored as part of the form definition. When you have created a script using this tab, it becomes available in the *Action Script Expression Builder* and you can call it from any

form control event. These scripts have a page-scope rather than an event-scope. For more information about using the script tab, see [Section 6.3.4, “Using the Scripts Tab,” on page 118](#).

The screenshot shows a 'Form Controls' dialog box. At the top, there are '+' and '-' icons. Below them is a table with three columns: 'Id', 'Type', and 'URL/Inline Script'. The first row has 'Script' in the 'Id' column and 'inline' in the 'Type' column. The 'URL/Inline Script' column is empty. There are four empty rows below the first one. At the bottom of the dialog, there are three tabs: 'Fields', 'Actions', and 'Scripts'. The 'Scripts' tab is currently selected.

Id	Type	URL/Inline Script
Script	inline	

6.3 Creating forms

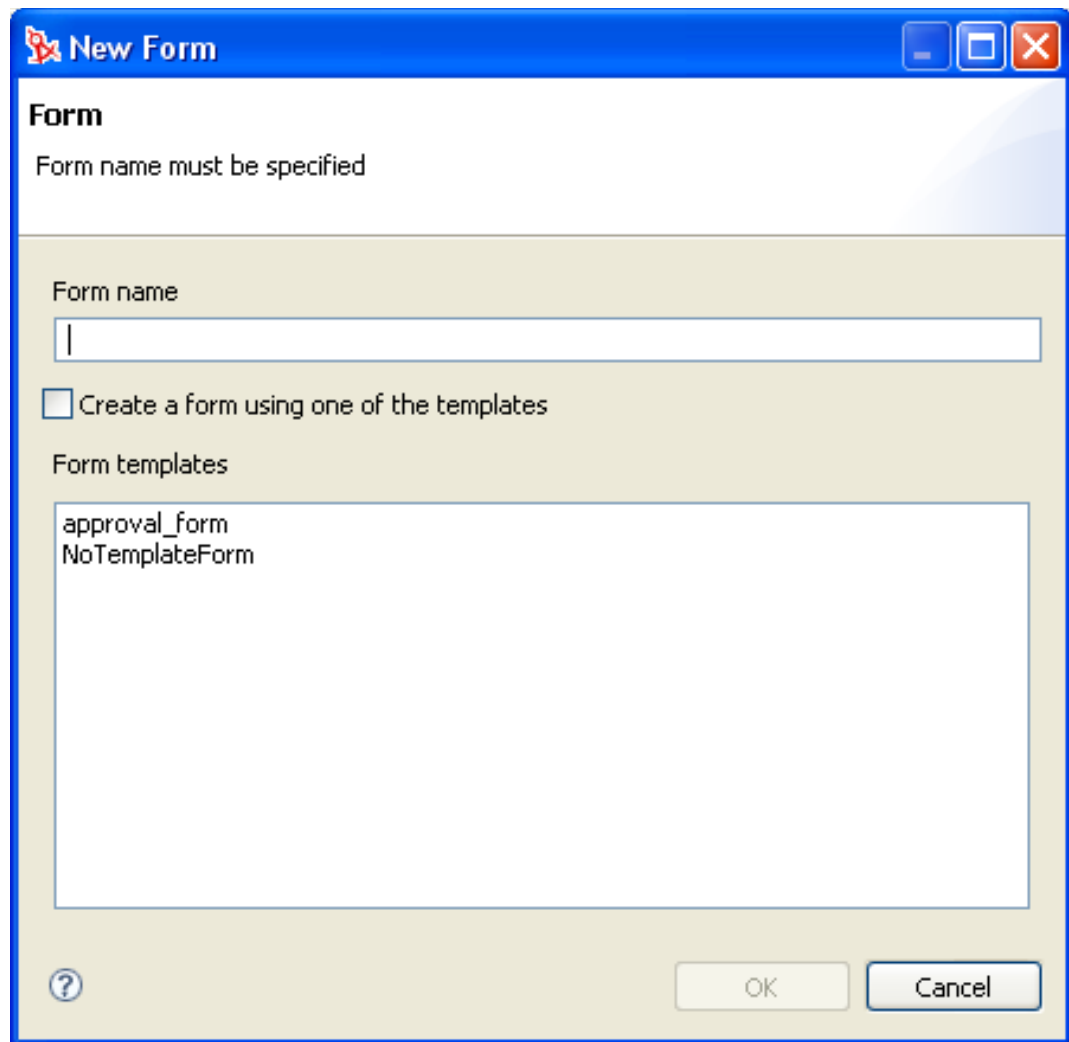
This section describes how to create new forms and add controls to it. It includes these topics:

- ♦ [Section 6.3.1, “Creating New Forms,” on page 111](#)
- ♦ [Section 6.3.2, “Adding Form Controls and Actions,” on page 112](#)
- ♦ [Section 6.3.3, “Defining Events,” on page 114](#)
- ♦ [Section 6.3.4, “Using the Scripts Tab,” on page 118](#)

6.3.1 Creating New Forms

- 1 With the provisioning request definition editor open, click the *Forms* tab.

- 2 In the Form Selection section of the page, click *Add* to access the New Form wizard.



- 3 Fill in the fields as follows:

Field	Description
<i>Form Name</i>	Type the name of the form as you want it to appear in Designer.
<i>Create a form using one of the templates</i>	If you want to base the new form on an existing template, select this option and select one of the forms from the Form templates list.

- 4 Click *OK* to save the form or *Cancel* to exit without saving.

6.3.2 Adding Form Controls and Actions

Use the *Form Controls* section to define the content and layout of the form.

To add a control to a form:

- 1 Click *Add*. Designer adds a control named `Field` to the bottom line of the form.

If you add more than one control of the same name to the form, Designer adds a unique number to the end of the control name.

- 2 Define the following properties for the control:

Field	Description
<i>Form Field Name</i>	A unique name for the field. The name is used in <ul style="list-style-type: none">♦ The <i>Workflow</i> tab's <i>Data Item Mapping</i> dialog box.♦ The <i>ECMA expression builder</i> dialog box.♦ An internal XML reference in the provisioning request definition file. <p>Consider the naming conventions you want to use for form fields to avoid confusion in the Data Item Mapping and ECMA expression builder dialog boxes. For example, the request and approval forms might both contain a field called <i>Reason</i>. To make it clear which field you are working with while performing data mappings, you can preface the field name with the name of the form where it is used. You might name one reason field <i>Req_Reason</i> and the other <i>App_Reason</i>.</p>
<i>Data Type</i>	The field's data type. The data type determines the valid control types and the type of validation performed.
<i>Control Type</i>	The type of visual control used to display or edit the data. The selection list is filtered based on the selected data type.
<i>Linebreaks</i>	Defines the number of lines you want inserted after the control.

NOTE: Form field controls do not have Data Item Mappings or E-mail notifications property sheets.

- 3 For each control, specify its properties in the *Properties* tab (available via *Window > Show View > Properties*). For more information, see [Section 6.5, "Form Control Reference," on page 122](#).
- 4 Click the *Actions* tab to define what the user can do with the form. For example, you can add actions that allow the user to submit a form or cancel it.

NOTE: A request form must have, at a minimum, a `SubmitAction`. Without a `SubmitAction`, the request will not process. It is recommended that every form also have a `CancelAction`. Each approval form must have at least one action defined.

- 5 In the *Actions* page, click *Add* to add a new *Action*. Fill in the fields as follows:

Field	Description
<i>Actions Location</i>	<p>Choose the location for the action buttons you add to the form.</p> <p><i>Bottom.</i> Places the action buttons on the bottom of the form. (Default.)</p> <p><i>Top:</i> Places the action buttons on the top of the form.</p> <p><i>Top and Bottom:</i> Places the buttons at both the top and bottom of the form.</p>
<i>Action Command</i>	<p>Choose an action for the button. For more information, see Section 6.4, "Action Reference," on page 120.</p>
<i>Linebreaks</i>	<p>Defines the number of lines you want inserted after the action button.</p>

Controlling Form Layout

The Designer places form controls on the form from top to bottom and left to right. Use Linebreaks to force spacing between controls.

6.3.3 Defining Events

The scripts you attach to an event handler are scoped to the appropriate control, not the browser window.

To define an event:

- 1 Select the form control where you want to define an event and open the property sheet.

- 2 Navigate to the Event property and add an event. Designer adds a row with the default event name *onEvent*.

[illegible]

- 3** Click in the *Event Name* field and select the *onchange* or *onload* event. For more information on adding other events, see **“Creating Custom Events” on page 115**.
- 4** Click in the *Action Expression* field. You can type the script directly in this field, or click the button to access the *Event Action Expression Builder*.
- 5** Define the action script, check the syntax, then click *OK*. Repeat this procedure to add more events to this control.

For more information on the `onChange` and `onLoad` events, see the events property description in [Section 6.5.2, “General Form Control Properties,” on page 124](#).

Creating Custom Events

You can create your own events to notify other controls of conditions or user actions on the form. You create the event using the *Events* property. You can give the event any name. You must explicitly fire the event using the **fireEvent()** method and passing in the name of the event.

You might want to perform a query on the Groups container that returns only the groups that match the values entered by a user. In the example shown in [Figure 6-7](#), the user types a value in the name

field, When the user tabs to the next field, the contents of the dropdown are populated from a query launched by the *namechange* custom event.

Figure 6-7 User Application Runtime Custom Event Sample

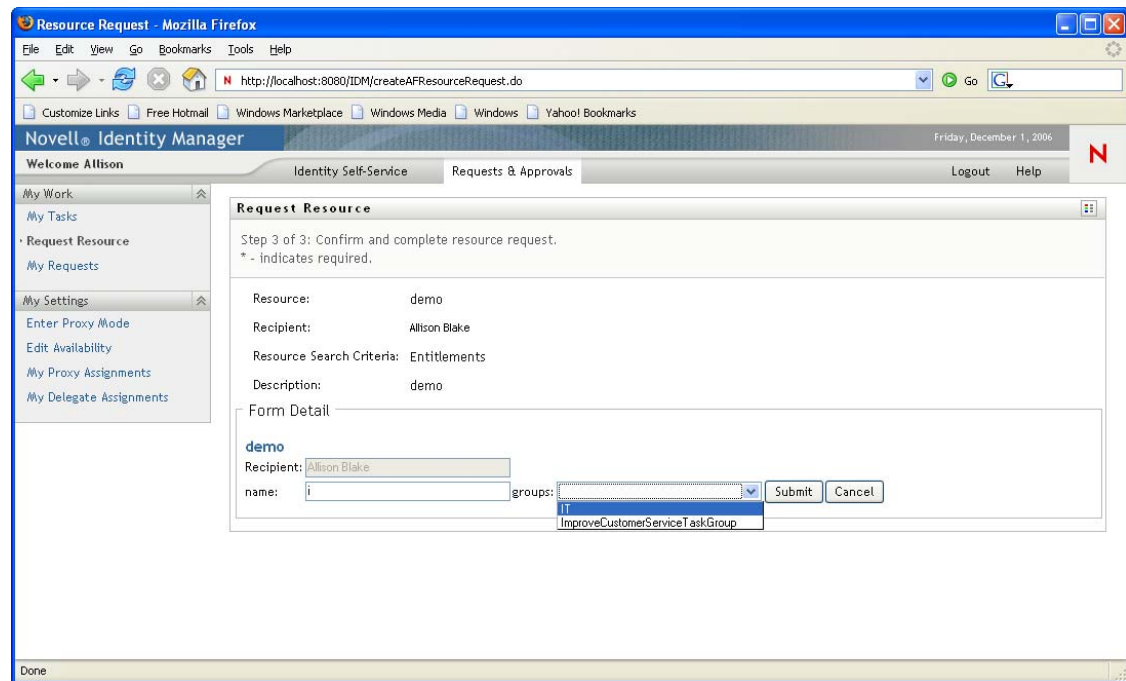
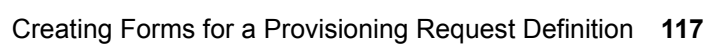
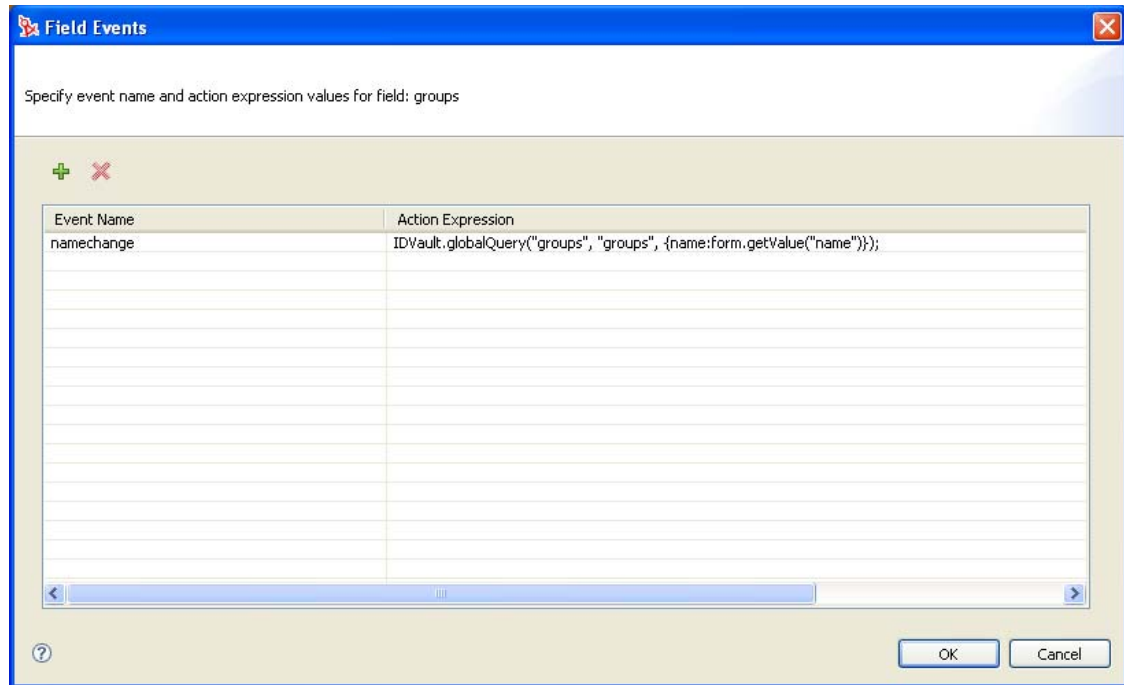


Figure 6-8 Sample `field.FireEvent()` Method



The *namechange* event contains an expression that executes a query called *groups*.

Figure 6-9 Custom Event Definition Example



For more information on using queries, see [Section 6.7, “Using DAL Queries in Forms,”](#) on [page 151](#).

6.3.4 Using the Scripts Tab


Use the *Scripts* tab to define a script that has a page-level scope. A page-level scope means that the script loads at page load time and is available through the life of the form. You can supply the script in one of the ways described in [Table 6-2](#).

Table 6-2 Script types

Script type	Description
<i>external</i>	The script is incorporated into the page by reference using the supplied URL. The script block will look something like this: <code><<script type="text/javascript" src="http://some.server/custom.js"></code> . The custom.js file is imported at form load.
<i>inline</i>	The script is inserted directly into the form in a <code><script></code> block.

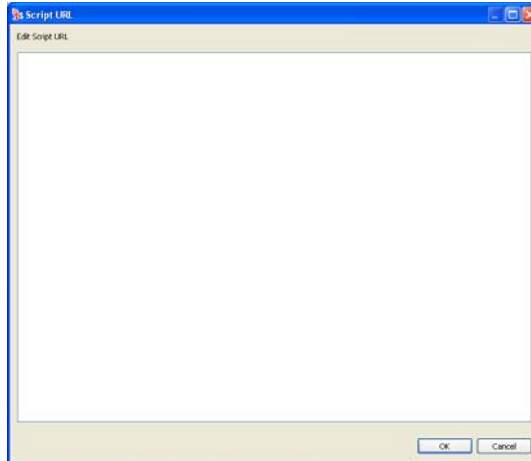
Because these scripts are loaded at page load, the form controls and any of their associated event handler scripts are not in scope when the page is loaded. Avoid coding dependencies between page-level scripts and event-level scripts; however, you can call page-level scripts from within an event-level script.

To add a link to an external JavaScript file:

- 1 With the Scripts tab open, click *Add* .

2 Complete the fields as follows:


Field	Description
<i>ID</i>	Specify a meaningful name. This value displays in the Event Action Expression Builder.
<i>Type</i>	Leave the default value of <i>external</i> .
<i>URL/Inline Script</i>	Click within the field so that the <i>Script URL Editor</i> button displays to the right, then click the button to display the editor.



Type the URL to the .js file and click *OK* to return to the *Scripts* tab. Adding an external link generates this line in the page that displays the form:

```
<script src="someURL.com/script.js"/>
```

To create an inline script:

- 1 With the *Scripts* tab open, click *Add* .
- 2 Complete the fields as follows:

Field	Description
<i>ID</i>	Specify a meaningful name. This value displays in the Event Action Expression Builder.
<i>Type</i>	Set the value to <i>inline</i> .
<i>URL/Inline Script</i>	<p>You edit your JavaScript using the <i>ECMAScript Editor</i>. You can access the editor by clicking the tab of the same name as the ID you specified above. This tab is located next to the <i>Signatures Declarations</i> tab.</p> <p>For inline scripts, the following is inserted in the page:</p> <pre><script>whatever you type</script></pre>

Both inline and external scripts are executed at page load but before the page loads the controls. In addition, they are also executed when specifically called on a form control event.

6.4 Action Reference

This section describes the actions you can add to forms. The actions are implemented as buttons. You can specify a custom display label for each button.

Table 6-3 *Valid Actions*

Action Name	Form Type	Description
<i>ApprovalAction</i>	Approval	Causes the Approval activity to complete and follow the <i>approved</i> flow path to the next activity. When you use this action, you must set the Hide If Read Only form property to True; otherwise the form fails validation when you deploy it. TIP: An ApprovalAction requires the Approval Activity associated with the form to have an approved flow path exiting the activity.
<i>CancelAction</i>	Request and Approval	For request forms, Cancel returns the user to the Request Resource Search Criteria form. For approval forms, Cancel returns the user to the My Tasks list.

Action Name	Form Type	Description
<i>CommentAction</i>	Approval forms	Generates a button with the default label set to <i>View Comment History</i> . The button launches a Comments dialog box displaying the processing history for each activity from the workflow start to the present time. Data displayed includes Date, Activity Name, User, and Comment as shown in the following example.

Date	Activity	User	Comments
06/01/2006 09:29:48 AM	Start	System	Workflow Started
06/01/2006 09:29:48 AM	Start	System	Workflow Forward
06/01/2006 09:30:18 AM	Single Approval	System	Workflow Claimed
06/01/2006 09:30:30 AM	Single Approval	Jack Miller	yes please

1 - 4 of 4

Refresh

Comments are updated and persisted to the workflow database through the UpdateAction (described below).

NOTE: Any forms containing this action must also contain a field named `apwaComment`.

<i>DenyAction</i>	Approval	<p>Causes the Approval activity to complete and follow the denied flow path. When you use this action, you must set the Hide If Read Only form property to True; otherwise, the form fails validation when you deploy it.</p> <p>TIP: A DenyAction requires the Approval Activity associated with the form to have a deny flow path exiting the activity.</p>
<i>RefusalAction</i>	Approval	<p>Causes the Approval activity to complete and follow the refused flow path. When you use this action, you must set the Hide If Read Only form property to True; otherwise, the form fails validation on deploy.</p> <p>TIP: A RefusalAction requires the Approval Activity associated with the form to have a refusal flow path exiting the activity.</p>
<i>SubmitAction</i>	Request and Approval	<p>Initiates the workflow and causes the workflow to execute the forward flow type. The workflow passes any user-entered data to the next activity in the workflow.</p>

Action Name	Form Type	Description
<i>UpdateAction</i>	Approval	Causes the Approval activity to write a user comment to the workflow database. There is typically a text area associated with an <code>apwaComment</code> form field. If the user enters text in this field and clicks this action, it is persisted to the <code>afcomment</code> table in the workflow database. The comment can be retrieved and viewed through the <code>CommentAction</code> (described above). The following example shows a text area and an update action button (labeled <code>UpdateAction</code>):

Single Approval

Please select the appropriate button to approve or reject the request.

Requested by: Allison Blake

Request Date: 06/05/2006

Reason: it is needed

Comment:

View Comment History

UpdateAction

Deny

Approve

NOTE: The form must contain a field named `apwaComment`; otherwise, the provisioning request definition fails validation.

For more information about `apwaComment`, see [“Controls for User-Entered Comments” on page 123](#).

The following table describes the properties you can set on actions.

Table 6-4 Action Properties

Property Name	Description
<i>Display Label</i>	Specifies the text to display on the button.
<i>Visible</i>	If true, specifies whether the action is visible at runtime.
<i>Block On Error</i>	If true, specifies that the action is blocked if any of the form's controls fail validation. This is recommended for the <code>SubmitAction</code> . Do not set to false if the action button submits data; otherwise, invalid data can be submitted causing unexpected results.
<i>Hide If Read Only</i>	If true, specifies that the action is hidden when the form is read-only. A form can be read-only when the user opens a task without claiming it first. If your form contains the <code>ApprovalAction</code> , <code>DenyAction</code> , or <code>RefusalAction</code> , this property must be set to true. If it is set to false, you will encounter a validation error and will not be able to deploy the form.

6.5 Form Control Reference

This section describes the controls you can add to a form.

Table 6-5 Control Types and Supported Data Types

Control Type	Data Types						
	Boolean	Date	Decimal	DN	Integer	String	Time
CheckBoxPickList	x		x	x	x	x	
DatePicker		x					
DateTimePicker							x
DNContainer				x			
DNDisplay				x			
DNLookup				x			
DNMaker				x			
DNQuery				x			
Global List						x	
Html						x	
MVCheckbox			x	x	x	x	
MVEditor			x	x	x	x	
PickList			x	x	x	x	
Static List	x		x		x	x	
Text			x	x	x	x	
Text Area						x	
Title						x	
TrueFalseRadioButtons	x						
TrueFalseSelectBox	x						

6.5.1 Controls for User-Entered Comments

Designer supports a special internal control you can add to a form to allow users to add comments to a workflow or to view previously entered comments. Comments are required on forms that use *CommentAction* or *UpdateAction*. The comments are not part of the workflow data so you cannot access them via the flowdata object. The comments are special data items stored in the afcomment table of the workflow database. The comments are persisted as long as the row for the requestid in the afprocess table exists.

To create a form that supports user comments:

- 1 Add a control to your form. Select Comment as the data type. The Form Field name is automatically defined as apwaComment and the Control Type is TextArea. A single form can contain only one comment field.
- 2 Add a *CommentAction* or *UpdateAction* to the form.

For more information, see [Section 6.4, “Action Reference,” on page 120](#).

6.5.2 General Form Control Properties

The properties in the following table are available for each control.

Table 6-6 *General Properties*

Property Name	Description
<i>Display label</i>	Specifies the label to display to identify the control. It is localizable.
<i>Editable</i>	Specifies if the control is editable (true). Otherwise, it displays as read-only.
<i>Events</i>	Specifies an event for the control. Possible values include the following: <ul style="list-style-type: none">◆ OnChange: Fires when one of the following occurs:<ul style="list-style-type: none">◆ Immediately after onload.◆ Another script changes the value of the control.◆ The user commits a change to the data value associated with the control. This occurs when the user has tabbed out of the control or otherwise caused it to lose focus. For example, this can happen when the user tabs away from the control (for text entry based controls like Text, TextArea, DatePicker), or when the user selects a different entry choice for choice based controls like PickList, MVCheckbox, and StaticList).◆ onLoad: The onload event for a control fires just once, when the control is loaded into the page for the first time. It can be used to set initial values or preselect entries; however, there is no guarantee that controls load in a particular order.
<i>Multivalued</i>	This is a read-only property. It specifies if the control supports multivalue attributes (true).
<i>Required</i>	Specifies whether the control requires user input (true).
<i>Tooltip</i>	Specifies the text for the control's tooltip. It is localizable.
<i>Visible</i>	Specifies whether the control is displayed in the user interface (true).

Sort Order

List-based controls sort content alphabetically. For DN-based lists, the sort order is alphabetical based on the Display expression property result. For all other types, the sort order is based on the display label.

6.5.3 CheckBoxPickList

Use the CheckBoxPickList control to allow users to view and choose one or more values from a dynamically generated list of choices displayed as checkboxes.

When the associated data type is a DN retrieved from the Identity Vault, you can display the checkbox label as the fully qualified DN or use the Display expression property to specify the attributes to display instead.

Figure 6-10 Sample CheckBoxPickList Control

CheckBoxPickList: ☐ Asia ☐ Europe ☐ North America ☐ South America

Table 6-7 CheckboxPickList Properties

Property Name	Description
<i>Entity Key for DN expression lookup</i>	<p>When you populate this control with a DN retrieved from the Identity Vault and you want that value to display in a user-friendly fashion, you should choose an entity from the drop-down list and specify a set of attributes in the Display expression property.</p> <p>Leave this value blank if you want to display the full DN or CN value retrieved from the Identity Vault.</p> <p>The entity you choose must have the directory abstraction layer View property set to true and be the entity whose DN you are retrieving from the Identity Vault.</p>
<i>Display expression</i>	<p>Required when you specify an <i>Entity Key for DN expression Lookup</i>. Choose the attributes to display as the checkbox labels. For example, to display the user entity's first and last name attributes, construct an expression like this: FirstName LastName.</p> <p>The attribute's directory abstraction layer properties for View, Read, Search, and Required must be set to true.</p>
<i>Allow multiple selections</i>	When set to true, users can select more than one entry.
<i>Sort entries</i>	When set to true, sorts results in ascending order. For details, see "Sort Order" on page 124 .

6.5.4 DatePicker

Use this control for display and entry of a date and time. This allows users to choose a date from a pop-up calendar or type a date in a text field. At runtime, the form automatically validates the date using the format for the user's locale and time zone. If the user enters an incorrect format, the form displays an error message. The DatePicker control's tooltip displays the valid date format. The default DatePicker control looks like this:

Figure 6-11 Sample DatePicker Control

DatePickerControl: 

When the Show date picker property is true, the form displays the date field along with a button. When the user clicks the button, the form launches a calendar for the user to select the date. The calendar pop-up is shown here:

Figure 6-12 *Sample Calendar Control*

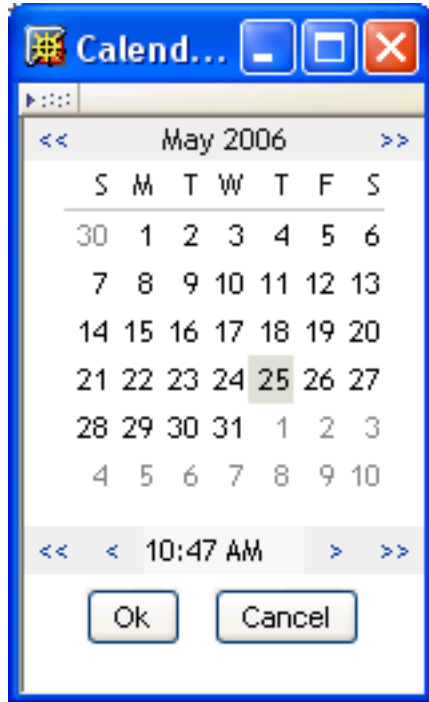


Table 6-8 *DatePicker Control Properties*


Property name	Description
<i>Datetime indicator</i>	When set to false, the Calendar pop-up does not display the time.
<i>Day headers</i>	A comma-separated, single-quoted list of values displayed by the Calendar pop-up to indicate the day of the week. This value is localizable.
<i>Field Width in pixels</i>	Use this field to configure the field's visible width on the form. The default is 200 pixels.
<i>Month names</i>	A comma-separated, single-quoted list of values displayed by the Calendar pop-up to indicate the month name. This value is localizable.
<i>Show date picker</i>	When set to true displays the calendar pop-up. If set to false, the calendar pop-up does not display. The user must type the date in the text field using the proper format for their locale.

6.5.5 DateTimePicker

Use this control for display and entry of a date and time for a Time data type. This allows users to choose a date and time from a pop-up calendar or type a the value in a text field. At runtime, the form automatically validates the date and time using the format for the user's locale and time zone.

If the user enters an incorrect format, the form displays an error message. The DateTimePicker tooltip displays the valid date format. The default control looks like this:

Figure 6-13 Sample DateTimePicker Control

DateTimePicker: 

When the *Show date picker* property is set to true, the form displays a text field followed by a calendar button. When the user clicks the calendar button, the form launches a calendar control for the user to select the date and time values. The calendar pop-up is shown here:

Figure 6-14 DateTimePicker Calendar Control

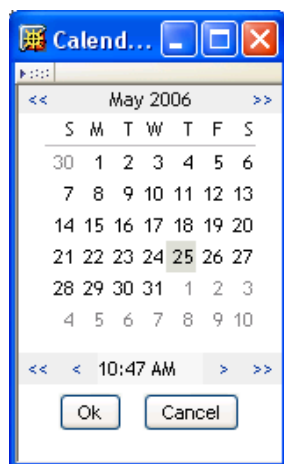


Table 6-9 DateTimePicker Control Properties

Property name	Description
<i>Day headers</i>	A comma-separated, single-quoted list of values displayed by the Calendar pop-up to indicate the day of the week. This value is localizable.
<i>Field width in pixels</i>	Use this field to configure the field's visible width on the form. The default is 200 pixels.
<i>isDateTime</i>	When set to false, the Calendar pop-up does not display the time.
<i>Month names</i>	A comma-separated, single-quoted list of values displayed by the Calendar pop-up to indicate the month name. This value is localizable.
<i>Show date picker</i>	When set to true, displays the calendar pop-up. If set to false, the calendar pop-up does not display. The user must type the date in the text field using the proper format for their locale.




6.5.6 DNContainer

Use this control to allow users to select a container object from within the root container that you specify. You can use this control to limit the user to a subtree of a container. This is a specialized version of the DNLookup control.

Figure 6-15 DNContainer Control With Root Container Specified



Table 6-10 DNContainer Control Properties

Property name	Description
Entity key used for object lookup	Choose an entity from the dropdown. The entity that you choose limits the users ability to look up objects within the specified entity's container. If you specify an entity key and a root container, the entity key takes precedence.
Field width in pixels	Use this field to configure the field's visible width on the form. The default is 200 pixels.
Root container	Specify a root container for lookups when users click the <i>Object Selector</i> button.
Show clear button	If set to true, the form displays the  <i>Reset field</i> button
Show object history button	If set to true, the form displays the  <i>Show history</i> button.
Show object selector button	If set to true, the form displays the  <i>Object Selector</i> button.

6.5.7 DNDisplay

Use this control to display a read-only DN. You populate the control from flowdata. The control can display the full DN or a set of attributes associated with the DN depending on the properties you set.

Figure 6-16 Sample DNDisplay

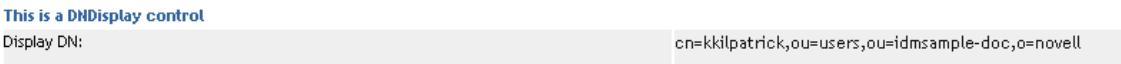


Figure 6-17 Sample DNDisplay with Display Expression Specified



Table 6-11 *DNDisplay Control Properties*

Property name	Description
<i>Display expression</i>	<p>Leave this value blank if you want to display the full DN or CN value.</p> <p>If you want to mask the DN by displaying attributes instead, launch the expression builder and select the desired attributes from the list. (You must first specify an <i>Entity key for DN expression lookup</i>.)</p> <p>For example, to show the user entity's first and last name attributes, construct an expression like this: <code>FirstName LastName</code>.</p> <p>Make sure the attribute's View, Read, Search, and Required properties are set to true in the directory abstraction layer. See Section 3.7.2, "Attribute Properties," on page 64.</p>
<i>Entity key for DN expression lookup</i>	<p>Leave this value blank if you want to display the full DN or CN value retrieved from the Identity Vault.</p> <p>If you want to mask the DN or CN by displaying attributes instead, choose the entity from the drop-down list and specify a set of attributes in the <i>Display expression</i> property.</p> <p>The entity you choose must</p> <ul style="list-style-type: none"> ◆ Have the directory abstraction layer View property set to true. ◆ Be the entity of the DN you are working with. <p>For more information, see Section 6.6, "Working with Distinguished Names," on page 147.</p>

6.5.8 DNLookup

Use this control to allow users to search and retrieve DNs from the Identity Vault. You can initialize the control with a DN from the flowdata. You set properties to control the entities and containers that the user can search and the format of the DN.

Figure 6-18 *Sample DNLookup Control*



The buttons associated with the DNLookup control are described in [Table 6-12](#).

Table 6-12 DNLookup Control Buttons


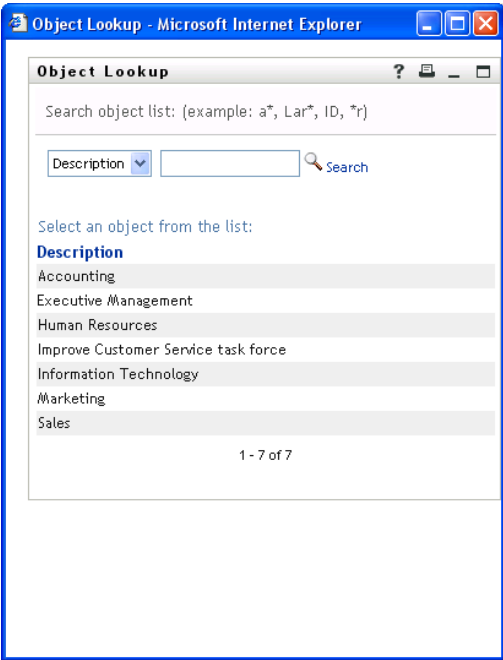



Button	Description
	<p>Launches an object lookup dialog box. You define whether the dialog box displays containers or objects via the <i>Object Selector type</i> property. The following is an example of an object lookup.</p> 
	<p>Show history. Allows users to view the history of objects that they have searched. They can select from this list or clear its contents. The availability of this button is controlled by the Show object history button property.</p>
	<p>Reset field. Deletes the field contents. The availability of this button is controlled by the Show clear button property.</p>

Table 6-13 *DNLookup Control Properties*

Property Name	Description
<i>Display expression</i>	<p>This property only applies when you initialize the control from flowdata. Leave this value blank if you want to display the full DN or CN value.</p> <p>If you want to mask the DN by displaying attributes instead, launch the expression builder and select the desired attributes from the list. (You must first specify an <i>Entity key for DN expression lookup</i>.)</p> <p>For example, to show the user entity's first and last name attributes, construct an expression like this: <code>FirstName LastName</code>.</p> <p>Make sure the attribute's View, Read, Search, and Required properties are set to true in the directory abstraction layer. See Section 3.7.2, "Attribute Properties," on page 64.</p>
<i>Entity key for DN expression lookup</i>	<p>This property only applies when you initialize the control from flowdata. Leave this value blank if you want to display the full DN or CN value retrieved from the Identity Vault.</p> <p>If you want to mask the DN or CN by displaying attributes instead, choose the entity from the drop-down list, then specify a set of attributes in the <i>Display expression</i> property.</p> <p>The entity you choose must</p> <ul style="list-style-type: none">♦ Have the directory abstraction layer View property set to true.♦ Be the entity of the DN you are working with. <p>For more information, Section 6.6, "Working with Distinguished Names," on page 147.</p>

Property Name	Description
<i>Object Selector type</i>	<p>Determines whether the object selector dialog box performs an Object Lookup or a Container Lookup. The following is an example of an Object Lookup:</p> 
	<p><i>paramlist</i>: Causes the object selector dialog to perform an object lookup. You specify the lookup criteria via the <i>Entity key used for object lookup</i> property.</p> <p><i>container</i>: Causes the object selector dialog to display one or more containers for selection. The containers for searching are determined by the <i>Search container</i> property, which is specified in the directory abstraction layer for the entity named in the <i>Entity key used for object lookup</i> property. For example, if the <i>Entity key used for object lookup</i> property is Group, the search container is set to %group-root% by default. If no search container is used, the search root specified during the User Application installation is used.</p>
<i>Field width in pixels</i>	Use this field to configure the field's visible width on the form. The default is 200 pixels.
<i>Show clear button</i>	If set to true, the form displays the <i>Reset field</i> button.
<i>Show object history button</i>	If set to true, the form displays the <i>Show history</i> button.

Property Name	Description
Show object selector button	If set to true, the form displays the <i>object selector</i> button.

6.5.9 DNMaker

Use this control to allow users to construct a DN value by specifying a naming value and choosing a container.

Figure 6-19 Sample DNMaker Control

Table 6-14 DNMaker Control Buttons


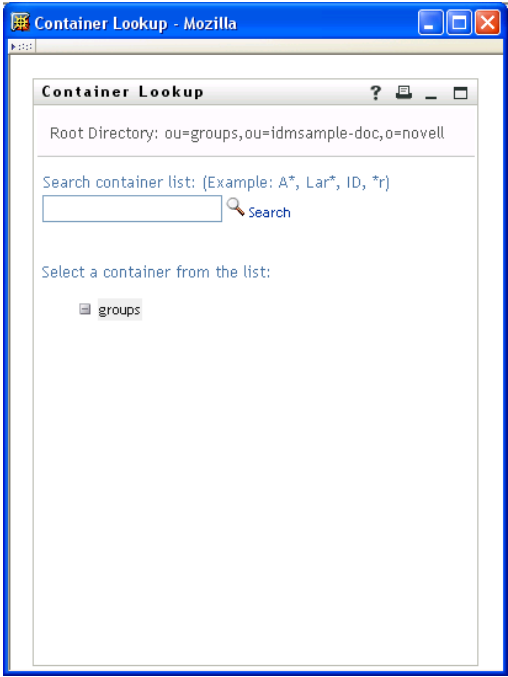


Button	Description
	Launches an object selector for container searches like the one shown below.
	
	The container search root is defined for the entity specified in the <i>Entity used for object lookup</i> property. The availability of this button is controlled by the Show object selector property.
	<i>Show history.</i> Allows users to view the history of objects that they have searched. They can select from this list or clear its contents. The availability of this button is controlled by the Show object history button property.
	<i>Reset field.</i> Deletes the field contents. The availability of this button is controlled by the Show clear button property.

Table 6-15 *DNMaker Control Properties*




Property	Description
<i>Entity key used for object lookup</i>	A required field. Choose an entity from the drop-down. This will determine the search that is launched when the user clicks the object selector button. If you specify an entity key and a root container, the entity key takes precedence
<i>Naming attribute</i>	The naming attribute used to construct the final DN. This value displays next to the control's display label as an extra hint to the user.
<i>Root container</i>	Specify a root container for lookups when users click the object selector button. If you do not specify a Root container, the User Application uses the container for the entity in the directory abstraction layer property called Search Container. If a search container is not specified for the specified entity, then the Root Container DN specified during the User Application installation is used. If you specify an entity key and a root container, the entity key takes precedence.
<i>Show clear button</i>	If set to true, the form displays the Reset field button.
<i>Show object history button</i>	If set to true, the form displays the Show history button.
<i>Show object selector button</i>	If set to true, the form displays the object selector button.

6.5.10 DNQuery

DNQuery is a specialized version of the DNLookup control. Like DNLookup, DNQuery allows users to search and retrieve DNs from the Identity Vault; however, with the DNQuery, the object selector content can be driven by the result of a directory abstraction layer Queries object rather than from properties.

Table 6-16 *DNQuery Control Properties*

Property name	Description
<i>DAL global query key</i>	Specifies the key of the DAL Queries object you want executed. You can select it from the Event Action Expression Builder. For more information about using DAL queries, see Section 6.7, "Using DAL Queries in Forms," on page 151 . For more information about defining DAL queries, see Section 3.4, "Working with Queries," on page 53 .
<i>DAL global query parameter(s)</i>	Specifies the value for the query parameters. For example, this passes the String Sales to the Queries parameter called groupname: <pre>(function () {return {"groupname": "Sales"}}) ();</pre>
<i>Display expression</i>	When you populate the control with initial data from a Data Item Mapping value, use this property to specify the attributes to display.

Property name	Description
<i>Entity key for DN expression lookup</i>	<p>This property only applies when you initialize the control from flowdata. Leave this value blank if you want to display the full DN or CN value retrieved from the Identity Vault.</p> <p>If you want to mask the DN or CN by displaying attributes instead, choose the entity from the drop-down list, then specify a set of attributes in the <i>Display expression</i> property.</p> <p>The entity you choose must</p> <ul style="list-style-type: none"> ◆ Have the directory abstraction layer View property set to true. ◆ Be the entity of the DN you are working with. <p>For more information, Section 6.6, “Working with Distinguished Names,” on page 147.</p>
<i>Field width in pixels</i>	Use this field to configure the field’s visible width on the form. The default is 200 pixels.
<i>Show clear button</i>	If set to true, the form displays the  <i>Reset field</i> button
<i>Show object history button</i>	If set to true, the form displays the  <i>Show history</i> button. When a DAL global query is specified, the object history button is not shown.
<i>Show object selector button</i>	If set to true, the form displays the  <i>Object Selector</i> button.

6.5.11 Global List

Use this control to allow users to select a single entry from a drop-down list. The contents of the list are defined in a directory abstraction layer global list element.

Figure 6-20 Sample Global List Control



Table 6-17 Global List Properties

Property Name	Description
<i>DAL global list key</i>	Specifies the unique identifier of the global list. This must correspond to the key specified in the directory abstraction layer.

For more information about global lists, see [Section 3.3, “Working with Lists,” on page 49](#).

6.5.12 Html

Use this control to add HTML fragments to the Form Detail. You can do this by specifying the HTML fragments in the HTML content property. In addition, you can conditionally add the HTML fragment via an event on the form control. In either case, specify the HTML through the use of an anonymous function, such as: `(function() { return "<yourTag yourAttr='your attr value' />"; }) ()`;

For example:

```
(function(){ return "<table bgcolor='#C0C0C0'><th colspan='3' align='center'>Table Header Goes Here</th><tr><td>Value 1.1</td><td>Value 1.2</td></tr><tr><td>Value 2.1</td><td>Value 2.2</td></tr></table>"; }) ()
```

6.5.13 MVCheckbox

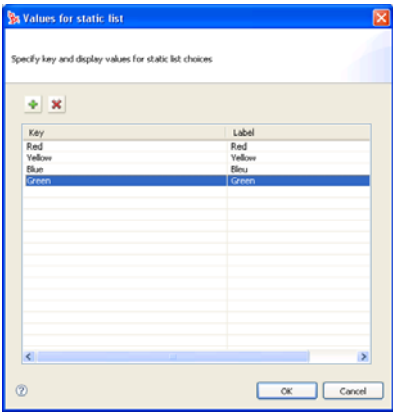
Use this control to display a set of labelled check boxes. You specify the label and its associated values through the List item property. A sample MVCheckbox control is shown below.

Figure 6-21 Sample MVCheckbox Control

MVCheckboxControl: ☐ Blue ☐ Green ☐ Red ☐ Yellow

Table 6-18 MVCheckbox Control Properties

Property Name	Description
List item	Allows you to define a set of static values that comprise the check box labels and values. Click the List property button to launch the list value dialog box shown here:



TIP: To retrieve user-entered values for this control, use `flowdata.getObject()` and not `flowdata.get()`. If you use `flowdata.get()`, you get only the first value.

For more information on preselecting values, see the [Section 10.2.3, “Form Control Examples,” on page 260](#).

6.5.14 MVEditor

Use this control to allow users to display, edit, or add multiple values in a drop-down list box. You can load the data dynamically from the Identity Vault, or allow users to enter the values.


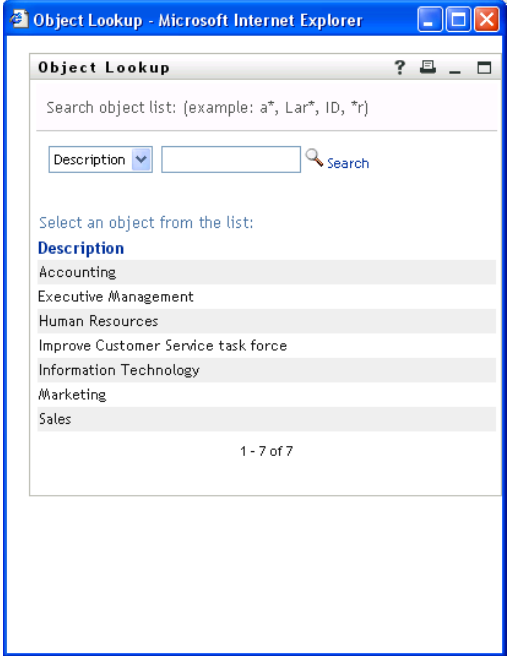


The control's appearance varies depending on the data type of the control and the properties that you specify. For example, if the data type is a DN, you can set properties that displays specific attributes related to the DN. You can also enable an object selector button that allows users to search and select values by setting the *Entity key used for object lookup* property.

There are also properties that let you specify a DAL Global Query to execute or specify a root DN to drive the object picker.

Figure 6-22 Sample MVEditor with Object Lookup Properties Set



Table 6-19 MVEditor with Object Selector Properties Set Control Buttons




Button	Description
	<p>Launches a search dialog box called an object selector. The object selector dialog box looks like this:</p>  <p>The user can select a value from the list to populate the control. The attribute displayed in the drop-down list (<i>Description</i> in the above example) is specified in the directory abstraction layer. You specify it in the attribute's UIControl property. See "Attribute UI Control Properties" on page 66. The availability of this button is controlled by the <i>Show object selector</i> property.</p>
	<p>Show history. Allows users to view the history of objects that they have searched. They can select from this list or clear its contents. The availability of this button is controlled by the <i>Show object history button</i> property.</p>
	<p>Reset field. Deletes the field contents. The availability of this button is controlled by the <i>Show clear button</i> property.</p>

If you do not set the object lookup properties, the MVEditor displays a simple edit control.

Figure 6-23 Sample MVEditor without Object Lookup Properties Set



Table 6-20 *MVEditor Control Buttons*

Button	Description
	Adds an item to the end of the list.
	Deletes the selected list item.
	Edits the selected list item.

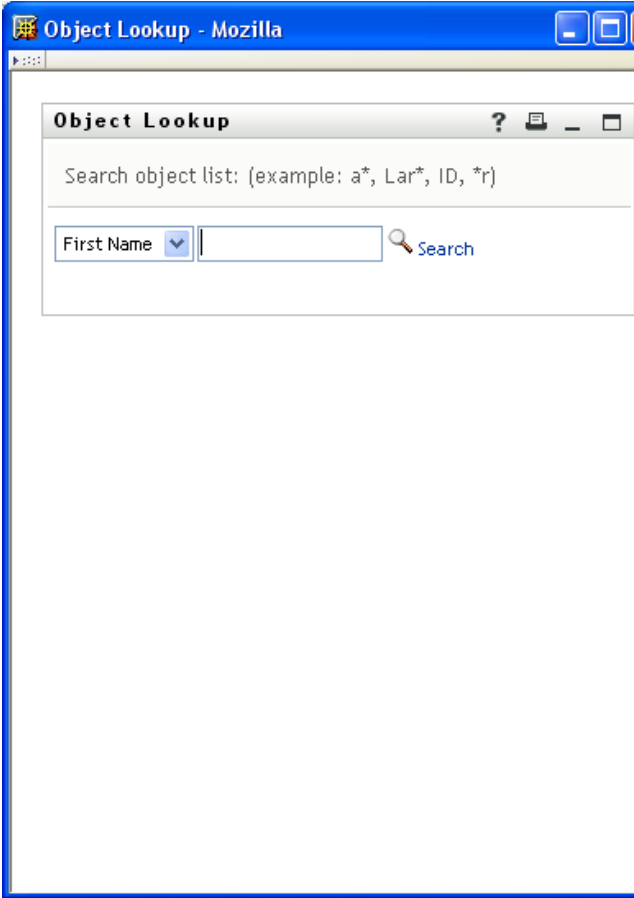
TIP: When the MVEditor control's Editable property is set to false, this control is read-only and the form does not display any MVEditor control buttons.

Table 6-21 *MVEditor Control Properties*

Property Name	Description
<i>Add data entry text field</i>	When set to true and there is a single row of data (and the data is not a DN), the control displays a data entry text field. The text field is displayed when the field is empty or contains only one value. Otherwise, the drop-down is displayed. If more than one row of data exists, then the drop-down always displays.
<i>DAL Global Query</i>	Specify this value if you want the control populated by the results of the Global Query that you specify. You specify the key name. You can select it from the Event Action Expression Builder. For more information about using queries in forms, see Section 6.7, "Using DAL Queries in Forms," on page 151 . For information about defining queries, see Section 3.4, "Working with Queries," on page 53 .
<i>DAL Global Query Parameter(s)</i>	Specifies the value for the query parameters. For example, this passes the String Sales to the queries parameter called groupname. <pre>(function () {return {"groupname": "Sales"}}) ();</pre>

Property Name	Description
<i>Display expression</i>	<p>Leave this value blank if you want to display the full DN or CN value.</p> <p>If you want to mask the DN or CN by displaying attributes instead, launch the expression builder and select the desired attributes from the list. (You must first specify an <i>Entity key for DN expression lookup</i>.)</p> <p>For example, to show the user entity's first and last name attributes, construct an expression like this: <code>FirstName LastName</code>.</p> <p>Make sure the attribute's View, Read, Search, and Required properties are set to true in the directory abstraction layer. See Section 3.7.2, "Attribute Properties," on page 64.</p>
<i>Enforce uniqueness</i>	Forces user-entered list items to be unique.
<i>Entity key for DN expression lookup</i>	<p>Leave this value blank if you want to display the full DN or CN value retrieved from the Identity Vault.</p> <p>If you want to mask the DN or CN by displaying attributes instead, choose the entity from the drop-down list and specify a set of attributes in the <i>Display expression</i> property.</p> <p>The entity you choose must</p> <ul style="list-style-type: none"> ◆ Have the directory abstraction layer View property set to true. ◆ Be the entity whose DN you are retrieving from the Identity Vault. <p>See Section 6.6, "Working with Distinguished Names," on page 147 for more information.</p>
<i>Field width in pixels</i>	Use this field to configure the field's visible width on the form. The default is 200 pixels.
<i>Ignore case</i>	If set to true, ignore case when enforcing uniqueness.
<i>Lower bound (for numbers only)</i>	Minimum integer or decimal value.
<i>Maximum length</i>	Maximum number of characters for string values. The control blocks input when this value is reached.
<i>Minimum length</i>	Minimum number of characters for string values. The control validates that the user enters at least this number of characters.

Property Name	Description
<i>Number of lines displayed</i>	<p>The number of lines displayed by the control. This is not the number of records retrieved or displayed, but the vertical size of the control. If you set this number to 10 and there are only 5 records to display, the control size is still 10 lines.</p> <p>You can set the number of lines to 1 or to 3 or greater. You cannot set it to 2 because it does not leave enough space for the browser to display scrollbars. If you set it to 2, Designer generates a warning in the Project Checker view and resets it to 3.</p>
<i>Numbers only</i>	If set to true, only numbers can be entered.

Property Name	Description
<i>Object Selector type</i>	<p>Determines whether the object selector dialog box performs an Object Lookup or a Container Lookup. The following is an example of an Object Lookup:</p> 
	<p><i>paramlist</i>: Causes the object selector dialog to perform an object lookup. You specify the lookup criteria via the <i>Entity key used for object lookup</i> property.</p> <p><i>container</i>: Causes the object selector to display one or more containers for selection. The containers for searching are determined by the <i>Search container</i> property, which is specified in the directory abstraction layer for the entity named in the <i>Entity key used for object lookup</i> property. For example, if the <i>Entity key used for object lookup</i> property is Group, the search container is set to %group-root% by default. If no search container is used, the search root specified during the User Application installation is used.</p>
<i>Resolve DN</i>	<p>When set to false, the DN is displayed rather than the Display expression. Consider using this when you expect a large number of DNs to be returned, and you are concerned about performance.</p>

Property Name	Description
<i>Root Container</i>	Specify a root container for lookups when users click the object selector button. If you specify an entity key and a root container, the entity key takes precedence.
<i>Show object history button</i>	When set to true, displays the <i>Object History</i> button next to the control.
<i>Show object selector button</i>	When set to true displays the <i>Object Selector</i> button next to the control.
<i>Sort entries</i>	When set to true, sorts the results in ascending order. For details, see “Sort Order” on page 124 .
<i>Upper bound (for Numbers only)</i>	The maximum numeric value users can enter.

TIP: To retrieve user-entered values for this control, use `flowdata.getObject()` and not `flowdata.get()`. If you use `flowdata.get()`, you get only the first value.

For more information about preselecting items, see [Chapter 10, “Working with ECMA Expressions,” on page 249](#).

6.5.15 PickList

Use the PickList control to allow users to view and choose one or more values from a dynamically generated list of choices. The list items are DN or CN values retrieved from the Identity Vault. You can display the full DN or CN or use the PickList properties to specify the attributes to display instead.

Figure 6-24 Sample PickList Control without DN Masking

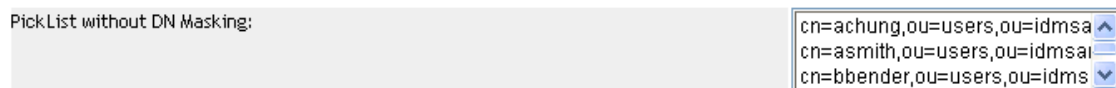


Figure 6-25 Sample PickList Control with DN Masking

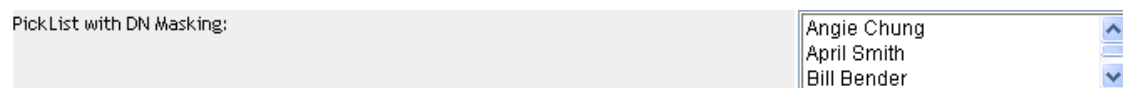


Table 6-22 PickList Control Properties

Property Name	Description
<i>Allow multiple selections</i>	<p>When set to true, the user can select more than one list value using their platform-specific multi-select keys.</p> <p>When set to true, the control displays a minimum of three lines regardless of the value specified in the Number of lines displayed property. If this value is false, the Number of lines displayed property is used.</p>

Property Name	Description
<i>Display expression</i>	<p>Leave this value blank if you want to display the full DN or CN value.</p> <p>If you want to format the DN or CN by displaying attributes instead, launch the expression builder and select the desired attributes from the list. (You must first specify an <i>Entity key for DN expression lookup</i>.)</p> <p>For example, to show the user entity's first and last name attributes, construct an expression like this: <code>FirstName LastName</code>.</p> <p>Make sure the attribute's View, Read, Search, and Required properties are set to true in the directory abstraction layer. See Section 3.7.2, "Attribute Properties," on page 64.</p>
<i>Entity key for DN expression lookup</i>	<p>Leave this value blank if you want to display the full DN or CN value retrieved from the Identity Vault.</p> <p>If you want to mask the DN or CN by displaying attributes instead, choose the entity from the drop-down list and specify a set of attributes in the <i>Display expression</i> property.</p> <p>The entity you choose must</p> <ul style="list-style-type: none"> ◆ Have the directory abstraction layer View property set to true. ◆ Be the entity whose DN you are retrieving from the Identity Vault.
<i>Field width in pixels</i>	<p>Use this field to configure the field's visible width on the form. The default is 200 pixels.</p>
<i>Number of lines displayed</i>	<p>The number of lines displayed by the control. This is not the number of records retrieved or displayed, but the vertical size of the control. If you set this number to 10 and there are only 5 records to display, the control size is still 10 lines.</p> <p>The number of lines displayed is related to the Allow multiple selections setting. When Allow multiple selections is set to true, the number of lines displayed is always 3 (or more). When Allow multiple selections is set to false, you can set the number of lines to 1 or to 3 or greater. You cannot set it to 2 because it does not leave enough space for the browser to display scrollbars. If you set it to 2, Designer generates a warning in the Project Checker view and resets it to 3.</p>
<i>Sort Entries</i>	<p>When set to true, sorts results in ascending order. For details, see "Sort Order" on page 124.</p>

TIP: To retrieve user-entered values for this control, use `flowdata.getObject()` and not `flowdata.get()`. If you use `flowdata.get()`, you get only the first value.

For more information on displaying the control with a preselected option, see [Section 10.2.3, “Form Control Examples,”](#) on page 260.

6.5.16 Static List

Use this control to display a list of items in a drop-down list from which users can select a single item. The list items are static and are stored with the provisioning request definition. The text “Click here to select” only appears if the field is not set to Required.

Figure 6-26 Sample Static List Control

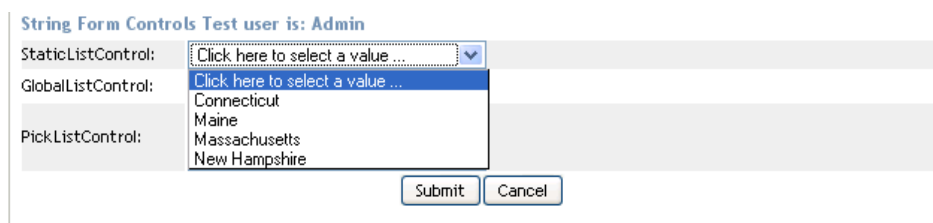
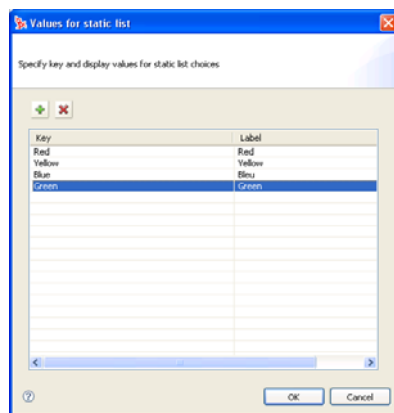


Table 6-23 Static List Properties

Property Name	Description
<i>Field width in pixels</i>	Use this field to configure the field’s visible width on the form. The default is 200 pixels.
<i>List item</i>	Allows you to define a set of labels and values that comprise the static list. Click the <i>List property</i> button to launch the list value dialog box shown here:



Click *Add* to add list items. Each list item must have a unique key. The dialog automatically generates a unique key when you insert a new list item. You can click the key name and change it. A blank key (null) is valid, so it is possible to have a list item with a blank key and a blank label. The displayed label is the one defined for the default language.

6.5.17 Text

Use the Text control for data display or user input. User input is validated depending on the control's data type.

Figure 6-27 Sample Text Control



Table 6-24 Text Control Properties

Property Name	Description
<i>Field width in pixels</i>	Use this field to configure the field's visible width on the form. The default is 200 pixels.
<i>Lower bounds (for numbers only)</i>	The lowest number allowed for decimal or integer values.
<i>Maximum length</i>	The maximum length for string values. Blocks input once this length is reached.
<i>Minimum length</i>	The minimum length for string values. Validates that the user enters a string at least this long.
<i>Number of characters allowed</i>	Specifies the number of characters a user is allowed to enter. This is related to Field width in pixels .
<i>Upper bound (for numbers only)</i>	The highest number allowed for decimal or integer values.
<i>Validation Mask (regular expression)</i>	An expression used for validating the field's data. Designer provides a default set of validation masks by default. You must enable them through <i>Windows > Preferences > Provisioning > Validation Mask</i> . For more information, see Section 2.3, "Setting Provisioning View Preferences," on page 24 .

6.5.18 Text Area

Use this control to display or accept input of multi-line data. Users can select multiple lines of data using the multi-select key combination for their platform.

Figure 6-28 Sample Text Area Control

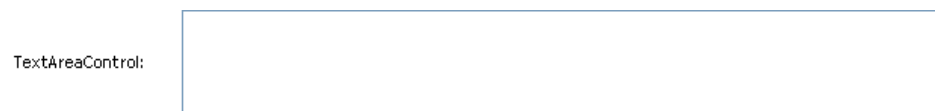


Table 6-25 Text Area Control Properties

Property Name	Description
<i>Number of columns displayed</i>	The visible width of the control; the number of characters wide.
<i>Number of lines displayed</i>	The number of lines to display at one time.

6.5.19 Title

Use this read-only control to label your form or provide instructions.

Table 6-26 *Title Control Properties*

Property Name	Description
<i>Display title in signed form document</i>	When set to false and the form is a signed form (using digital signatures), the title control is not displayed.
<i>Font-size</i>	Specify small, medium, or large.
<i>Style class</i>	Choose font style (such as bold) and colors from a palette.

6.5.20 TrueFalseRadioButtons

Use this control to display a choice of True or False as a set of radio buttons.

Figure 6-29 *Sample TrueFalseRadioButtons Control*

TrueFalseRadiobuttons: ☐ False ☒ True

This control has no custom properties.

6.5.21 TrueFalseSelectBox

Use this control to display a choice of True or False in a drop-down. The text “Click here to select a value” displays only when the field is not required.

Figure 6-30 *Sample TrueFalseSelectBox Control*

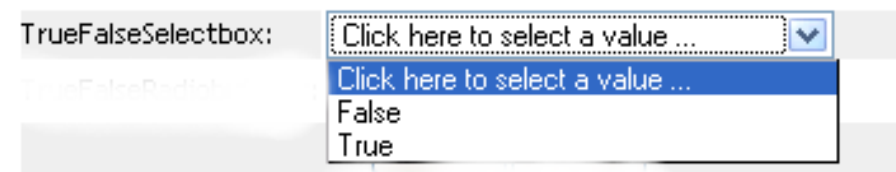


Table 6-27 *TrueFalsSelectBox Properties*

Property Name	Description
<i>Field width in pixels</i>	Use this field to configure the field's visible width on the form. The default is 200 pixels.

6.6 Working with Distinguished Names

The following controls provide specialized support for Distinguished Names (DNs):

- ♦ **DNDisplay**

- ◆ DNLookup
- ◆ DNMaker
- ◆ MVEditor
- ◆ PickList

This section describes the specialized support, including the following:

- ◆ Section 6.6.1, “Formatting DNs,” on page 148.
- ◆ Section 6.6.2, “Working with Object Selectors,” on page 148.

6.6.1 Formatting DNs

If you have a DN value, you can display either the DN or a set of attributes related to that DN. For example, if the control displays the DN of a user entity, you could display the user entity's First Name and Last Name attributes instead. The control's that support this feature are: DNDisplay, DNLookup, MVEditor, and Picklist.

You define the attributes to display in the control's Display Expression property. This display expression resolves at runtime by replacing the attribute keys with the attribute values.

6.6.2 Working with Object Selectors

In some cases, you might want the user to search for and select a DN from a list of possible values. The object selector dialog (also called the object lookup dialog) provides this functionality. The contents of the object selector dialog box are controlled by the form control's properties (see Table 6-28), and by how DAL properties are defined (see “DNLookup Control Type Definitions and Object Selector Contents” on page 149).

Table 6-28 Properties for Defining the Object Selector Dialog Box

Property	Description
<i>Entity key used for object lookup</i>	This is the key to the directory abstraction layer entity whose DN you want to search for or display. This is a required field.
<i>Object selector type</i>	<p><i>paramlist</i>: Causes the object selector dialog to perform an object lookup. You specify the lookup criteria via the <i>Entity key used for object lookup</i> property.</p> <p><i>container</i>: Causes the object selector dialog to display one or more containers for selection. The containers for searching are determined by the Search container property which is specified in the directory abstraction layer for the entity named in the <i>Entity key used for object lookup</i> property. For example, if the <i>Entity key used for object lookup</i> property is Group, the search container is set to <i>%group-root%</i> by default. If no search container is used, the search root specified during the User Application installation is used.</p>
<i>Show object selector button</i>	If set to true, the object selector button shows up on the control. Otherwise, it does not.

DNLookup Control Type Definitions and Object Selector Contents

When you specify an *Entity key used for object lookup*, the object selector's contents are defined by the attribute's DNLookup control type definition (in the directory abstraction layer). For example, if you specified the User entity as the object lookup and the manager as the attribute, the object selector would allow the user to search on the First Name and Last Name attributes because The object selector uses the manager's DNLookup control type definition to determine the lookup criteria. The DNLookup definition for the manager entity is shown in [Figure 6-31](#).

Figure 6-31 Manager Attribute on User DNLookup Property Definition

The screenshot shows a configuration window for a DNLookup control type. It is divided into two main sections: 'UI Control' and 'DNLookup Display'.

UI Control

Specify any formatting or special controls used in displaying the attribute:

Data Type:

Format Type:

Control Type:

DNLookup Display

Select the Entity and Attributes to display for the Lookup operation:

Lookup Entity:

Lookup Attributes

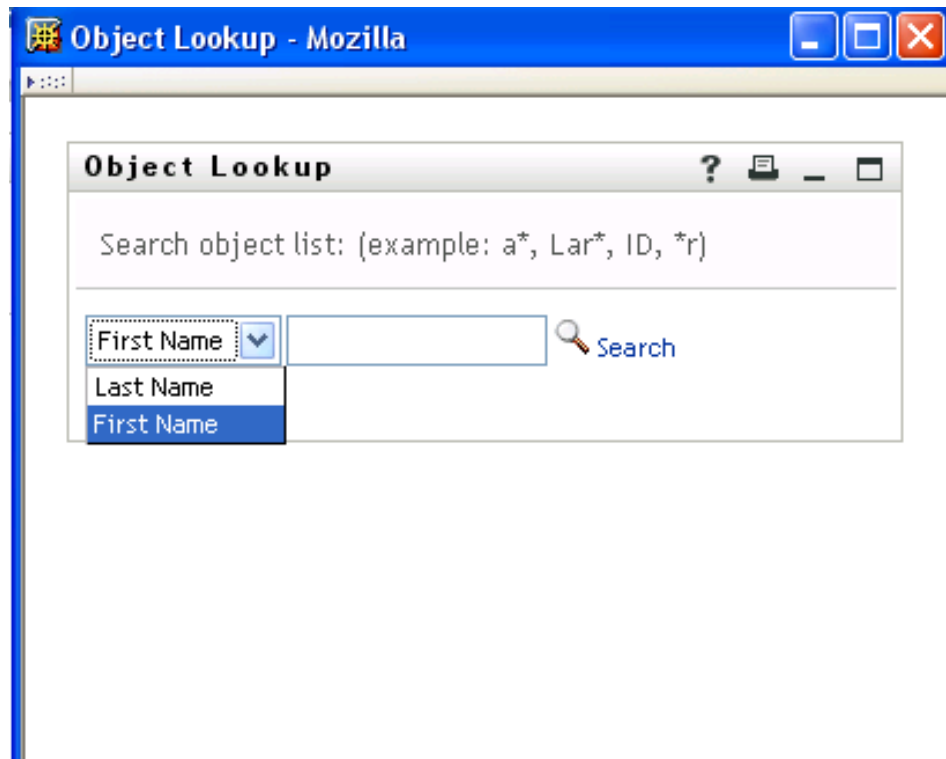
There are two attributes listed:

- Last Name (with a dropdown arrow and a red X icon)
- First Name (with a dropdown arrow and a red X icon)

☐ Perform Automatic Query

The resulting object selector is shown in [Figure 6-32](#).

Figure 6-32 Sample Object Selector



You can change the attributes that are used by the object selector by changing the Lookup attributes. To allow other attributes in the object selector:

- 1 Determine if the desired attribute is defined for the entity specified as the Lookup Entity. (In this example it is Manager Lookup.)
- 2 If the attribute you want is available on the lookup entity, you can just add it to the Lookup Attributes. Make sure that it has the Search and Read properties set to true; otherwise, they won't appear in the object selector dialog box.
- 3 If the attribute does not already exist for the Lookup Entity, you must do the following:
 - ♦ Add the attribute to the Lookup Entity. For example, to display another attribute in a manager lookup like the one above, add the attribute to the Manager Lookup entity. For more information, see [Section 3.2.3, "Adding Attributes," on page 47](#).
 - ♦ Add the attribute to the DNLookup definition.
 - ♦ Deploy the changed definitions. In this example, you'd redeploy the Manager Lookup entity (if you added a new attribute to its definition) and the User entity because you changed the definition of the manager attribute.
 - ♦ Refresh the application server's DirectoryAbstractionLayerDefinitions cache.

6.7 Using DAL Queries in Forms

The Query objects defined in the directory abstraction layer let you predefine LDAP searches that you can then execute from a workflow form. The information in this illustrates how you can define a query and use it in a form.

Suppose that you want to distribute calling cards to certain employees, but you only want to distribute calling cards to employees who work at home, and whose homes are located outside of the local office's area code. You create a workflow form that allows the employee to

- ♦ Verify that they qualify to receive a card.
- ♦ Submit a request for a card if they do qualify.

On your form, you allow the user to enter the area code of their own local office and based on that area code, review a list of users that qualify for a card. The runtime form is shown in [Figure 6-33](#).

Figure 6-33 Sample Calling Card Request Form

Resource Request - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://localhost:8080/IDM/createAFResourceRequest.do

Novell Identity Manager Friday, January 26, 2007

Welcome Bill Identity Self-Service Requests & Approvals Logout Help

My Work
My Tasks
Request Resource
My Requests
My Settings
Enter Proxy Mode
Edit Availability
My Proxy Assignments
My Delegate Assignments

Request Resource

Step 3 of 3: Confirm and complete resource request.
* - indicates required.

Resource: CallingCard
Recipient: Bill Brown
Resource Search Criteria: Accounts
Description: CallingCard

Form Detail

Get A Calling Card for Calls Outside Your Local Office Area Code
OfficeAreaCode: 212

The Calling Card is for Work At Home Employees Who Live Outside the Area Code of the Local Office

Candidates:
Alicon Blake (555) 555-1222
Angie Chung (555) 555-1208
Anthony Palani (555) 555-1202

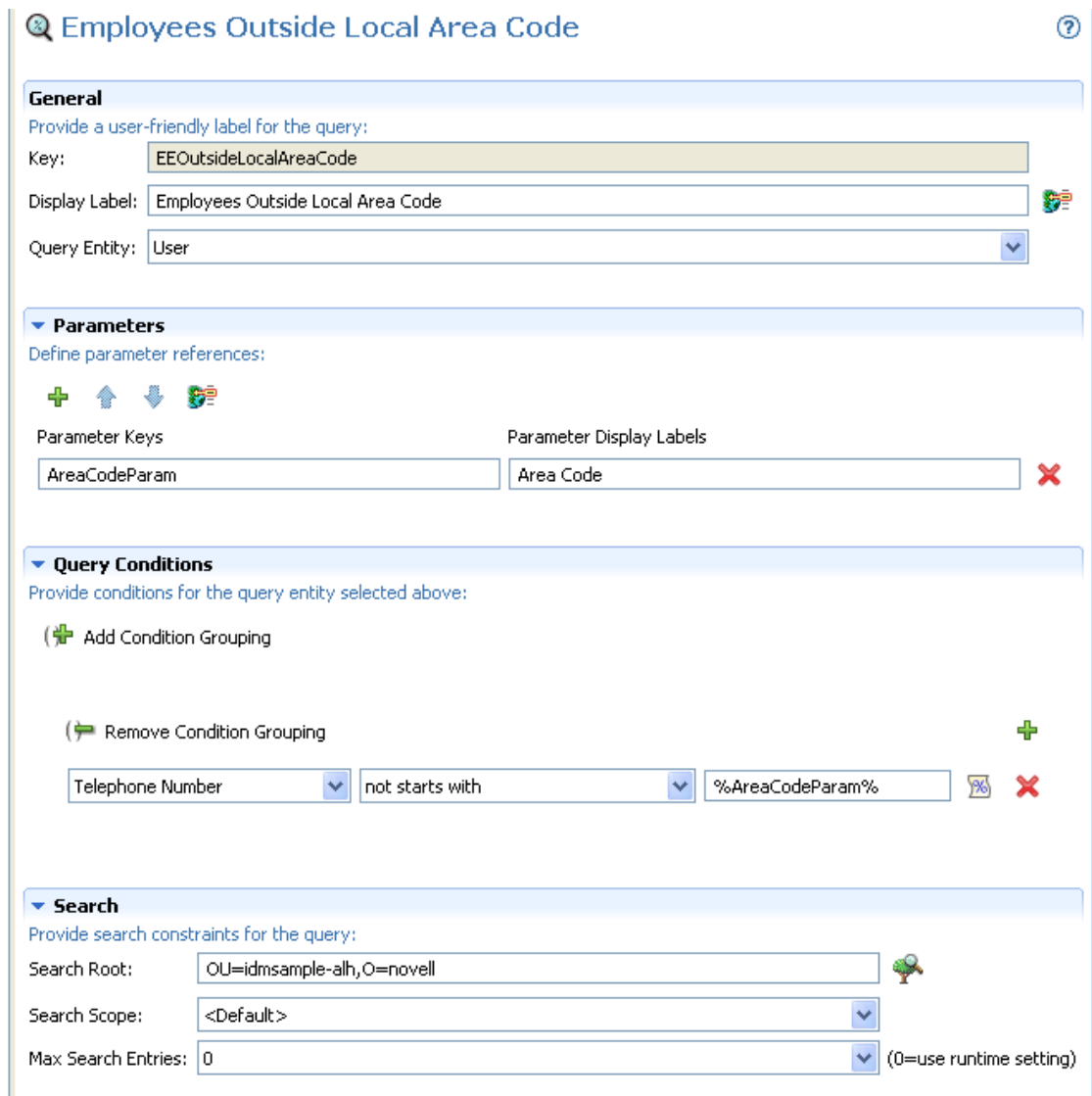
If you do not see your name in the above list you do not qualify for the calling card, please press Cancel.

Submit Cancel

Done

The data in the Candidates Picklist control is populated from the results of a query that is defined as shown in [Figure 6-34](#).

Figure 6-34 Calling Card Queries Definition



Employees Outside Local Area Code

General
Provide a user-friendly label for the query:
Key:
Display Label:
Query Entity:

Parameters
Define parameter references:
Parameter Keys:
Parameter Display Labels:

Query Conditions
Provide conditions for the query entity selected above:
(+) Add Condition Grouping
(-) Remove Condition Grouping
Telephone Number

Search
Provide search constraints for the query:
Search Root:
Search Scope:
Max Search Entries: (0=use runtime setting)

The query takes a single input parameter, *AreaCodeParam*, for the user-entered area code. The query then searches the *User* entity (in the *idmsample-alh* container) and returns the users whose telephone numbers do *not start with* the same value entered in the *AreaCodeParam*.

The form has an input field called *OfficeAreaCode*. It is the text field where the user enters the area code of the local office. The properties for OfficeAreaCode are show in [Figure 6-35](#).

Figure 6-35 *OfficeAreaCode Properties*

Form Controls

+
×
↑
↓

Form Field Name	Data Type	Control Type	Linebreaks
title	string	Title	1
OfficeAreaCode	string	Text	1
title2	string	Title	1
Candidates	dn	PickList	1
title3	string	Title	1

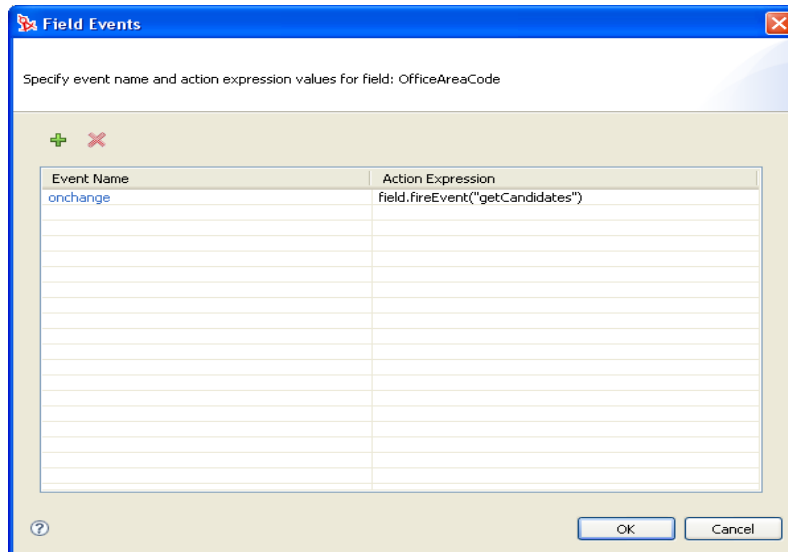
Outline ☒ Properties ☐

Property	Value
Display Label	OfficeAreaCode
Editable	true
Required	false
Visible	true
Multivalued	false
Events	onchange
Field width in pixels	200
lower bound (for Numbers only)	
Maximum length	3
Minimum length	3
Number of characters allowed	40
Tooltip	
Upper bound (for Numbers only)	
Validation Mask(regular expression)	

Fields / Actions / Scripts

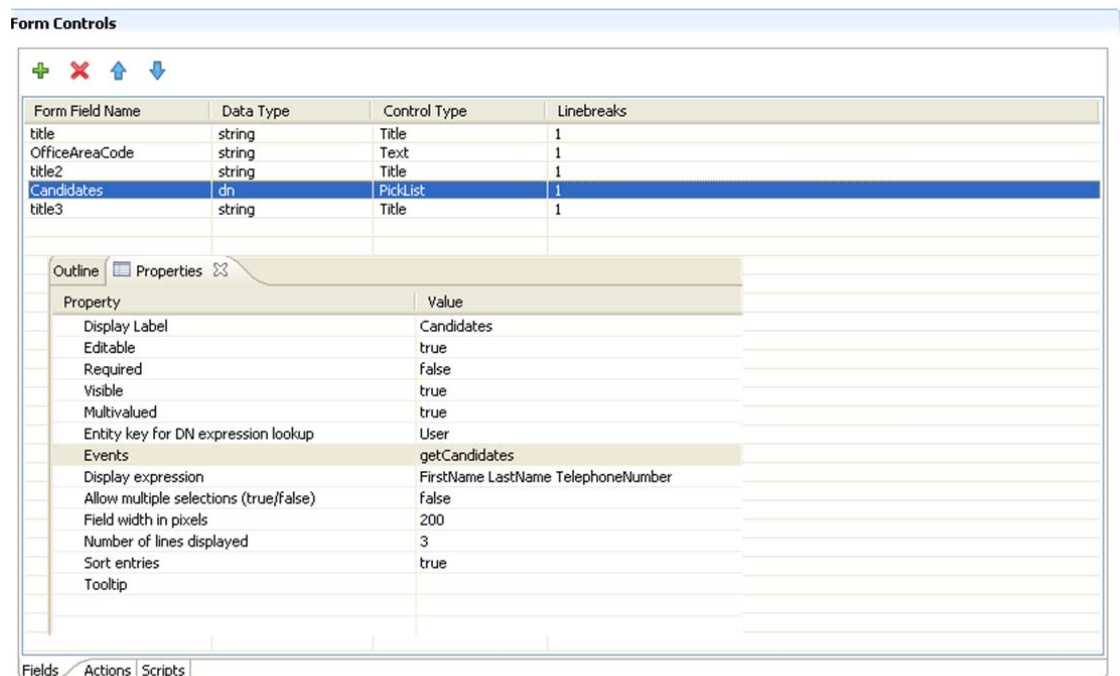
Notice that the Text control defines an *onchange* event. The *onchange* event fires when the user tabs from the Text control. The *onchange* event fires the *getCandidates* custom event as shown in [Figure 6-36](#).

Figure 6-36 OnChange Event Properties



The *getCandidates* event is defined as a property on the *Candidates* Picklist control.

Figure 6-37 Candidates PickList Properties



When the event is fired, the *getCandidates* event performs an action expression that calls the *globalQuery()* method (as shown in [Figure 6-38](#)). This method populates the value of the *Candidates* PickList control with the results of the query called *EEOutsideLocalAreaCode* (defined

Figure 6-38 *GetCandidates Event*

Creating Forms for a Provisioning Request Definition 155

Creating the Workflow for a Provisioning Request Definition

7

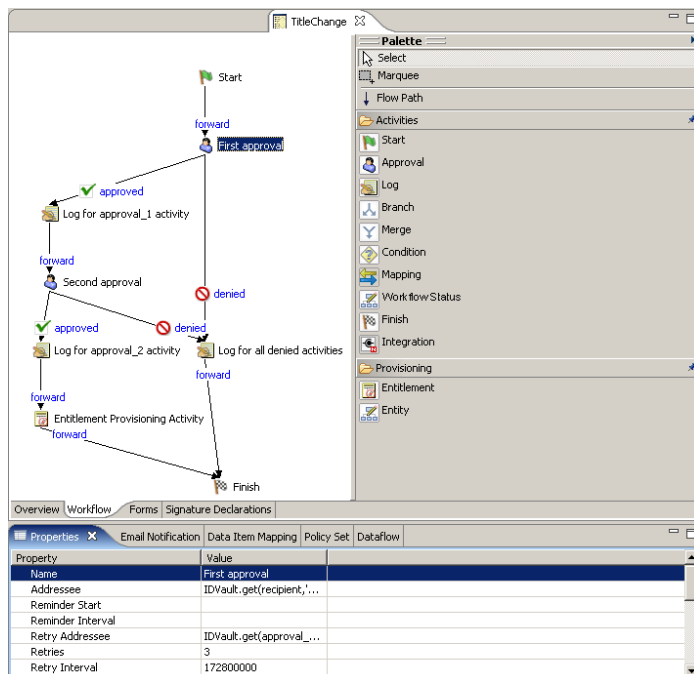
This section provides details on creating the workflow for a provisioning request definition. Topics include:

- ♦ Section 7.1, “About the Workflow Tab,” on page 157
- ♦ Section 7.2, “Adding Activities to a Workflow,” on page 161
- ♦ Section 7.3, “Adding the Flow Paths,” on page 164
- ♦ Section 7.4, “Configuring Flow Paths,” on page 165
- ♦ Section 7.5, “Addressing an Approval Activity,” on page 167
- ♦ Section 7.6, “Provisioning Multiple Individuals with One Workflow Instance,” on page 171
- ♦ Section 7.7, “Working with Entity Activities,” on page 173
- ♦ Section 7.8, “Configuring Digital Signature Support,” on page 175

7.1 About the Workflow Tab

You use the *Workflow* tab to display the Workflow page. You use the Workflow page to define the behavior of the workflow for the provisioning request definition. The Workflow page consists of a canvas, a palette, and associated views.

Figure 7-1 Workflow Page



7.1.1 Canvas

The canvas provides a graphical view of the activities in the workflow. When you create a new provisioning request definition that is not based on a template, the canvas is blank except for a Start and Finish activity.

If you right-click anywhere on the canvas, a menu is displayed. The menu includes the following commands:

Table 7-1 Workflow Menu

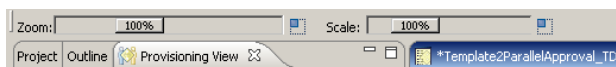
Item	Description
Delete	Deletes the selected activity or flow path.
Show Activity IDs	Switches the workflow editor between displaying activity names and activity IDs. Activity IDs are system defined and are not editable. However, if errors associated with activities are detected during validation, Designer identifies the activity in which the error occurred by activity ID. When this is the case, turn on the display of activity IDs in order to locate the activity on the canvas. You can specify whether activity names or activity IDs are displayed by default by choosing <i>Window > Preferences > Provisioning > Workflows > Diagram Preferences > Show Activity IDs</i> .
Show Flow Path Types	Turns the display of flow path types (for example, forward, approved, denied) on and off. When <i>Show Flow Path Types</i> is turned on, a label is displayed on each flow path indicating the flow path type.
Show Properties	Displays the Properties view for the selected activity.
Show Data Item Mapping	Displays the Data Item Mapping view for the selected activity.
Show Email Notification	Displays the Email Notification view for the selected activity.

The canvas provides two controls that make it easier to view the workflow:

Zoom: Use the *Zoom* control to increase or decrease the magnification of the workflow display. You can make portions of the workflow display larger, to view more detail, or make the workflow display smaller, to view more of the workflow. Click the rectangle to the right of the *Zoom* control to return to 100% magnification.

Scale: Use the *Scale* control to increase or decrease the spacing between items in the workflow display. For example, if your workflow has items with many flowpaths between them, you can increase the scale to make it easier to see individual flow paths. Click the rectangle to the right of the *Scale* control to return to 100% scale.

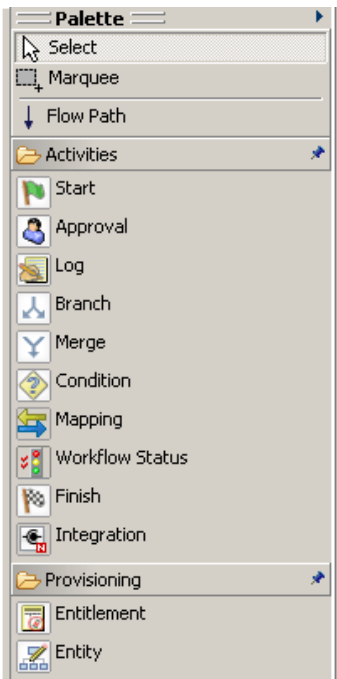
Figure 7-2 Zoom and Scale Controls



7.1.2 Palette

The palette provides icons for activities that can be dragged onto the canvas to create the workflow. It also provides tools for manipulating the icons and for linking activities:

Figure 7-3 Workflow Palette



The palette includes the following tools:

Table 7-2 Workflow Palette

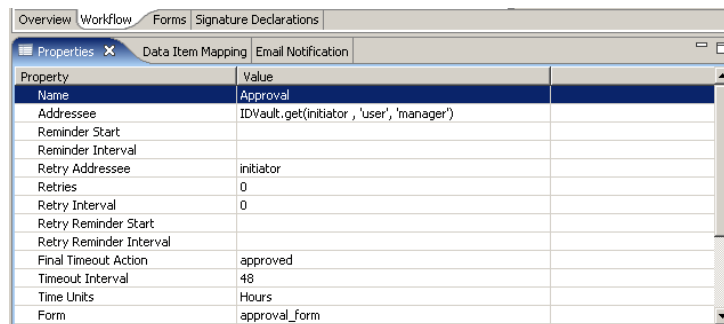
Tool	Description
Select	Selects individual nodes or flow paths. To select a node, click the Select tool, then click a node.
Marquee	Selects multiple nodes or flow paths. Use this tool to move items as a group. To select multiple items, click the Marquee tool, then click in an area outside of the items that you want to select. Hold down the mouse button and drag over the items that you want to select, then release the mouse button. When multiple items are selected, only the properties for the first item selected are displayed in the Properties view (see Section 7.1.3, “Views,” on page 160 for information about Views).
Flow Path	Creates flow paths between nodes. Flow paths provide connection logic for connecting nodes. For information about connecting nodes, see Section 7.3, “Adding the Flow Paths,” on page 164 .

Tool	Description
Activities (for example, Start, Approval, Log)	Inserts the selected activity into the workflow. For information about adding activities, see Section 7.2, “Adding Activities to a Workflow,” on page 161 . For detailed descriptions of the activities, see Chapter 8, “Workflow Activity Reference,” on page 179 .

7.1.3 Views

The Workflow page also includes the Properties, Data Item Mapping, and Email Notification views:

Figure 7-4 Workflow Views



You can right-click the icon for an activity to select a view from a context menu. Not all activities utilize all views. The following table identifies the views and the activities that use them:

Table 7-3 Views for Activities

Activity	Properties	Email Notification	Data Item Mapping
Start	X		X
Approval	X	X	X
Log	X		
Branch	X		
Merge	X		
Condition	X		
Mapping	X		X
Workflow Status	X		X
Finish	X	X	
Integration	X		X
Entitlement	X		X
Entity	X		X

7.2 Adding Activities to a Workflow

- 1 Click the *Workflow* tab. A graphical representation of the workflow for the provisioning request definition is displayed:



Because every workflow must have a Start and Finish activity, these activities are added to the canvas automatically. The Start Activity is connected to the Finish Activity with a forward link.

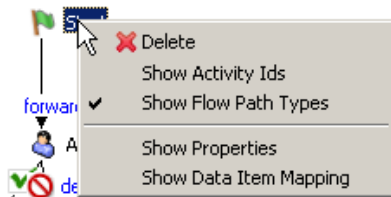
- 2 To add an activity to the workflow, click the icon for the desired activity in the palette and drag the icon onto the workspace.

You can insert an activity between activities that are linked by a flow path by dropping the activity onto the flow path. For information about defining flow paths between activities, see [Section 7.3, “Adding the Flow Paths,” on page 164](#). After you have added an activity to the workflow, you should set the properties of the activity (see [Section 7.2.1, “Setting the General Properties of an Activity,” on page 161](#)). For detailed information about configuring the different types of activities, see [Chapter 8, “Workflow Activity Reference,” on page 179](#) and [Chapter 9, “Working with Integration Activities,” on page 201](#).

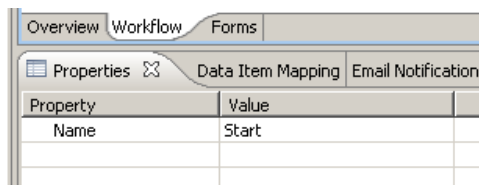
7.2.1 Setting the General Properties of an Activity

- 1 Right-click the activity icon for which you want to set properties and select *Show Properties* from the menu.

TIP: You can also display the *Properties* tab by selecting *Show Properties* from the *PRD* menu.



The Properties view is displayed:



- 2 Click in the column for a property to set the property. For information about the properties for each activity, see [Chapter 8, “Workflow Activity Reference,” on page 179](#).

Each activity has a default name. We strongly recommend that you replace the default names of activities with names that describe the specific purpose of the activity in the workflow. This makes it easier to understand the workflow when you look at the graphical display of the workflow. It also makes comments displayed in the User Application easier to understand. For example, the following figures show comments in the User Application using default IDs and descriptive IDs.

Figure 7-5 Activities in User Comments Using Default Names

Process Comments			
Date	Activity	User	Comments
01/04/2007 04:55:54 PM	First approval	IDMProv	User task assigned to reviewer Margo MacKenzie
01/04/2007 04:56:27 PM	First approval	IDMProv	User task claimed by reviewer Margo MacKenzie
01/04/2007 04:56:33 PM	First approval	IDMProv	User task approved by reviewer Margo MacKenzie
01/04/2007 04:56:33 PM	Second approval	IDMProv	User task assigned to reviewer Human Resources
01/04/2007 04:56:58 PM	Second approval	IDMProv	User task claimed by reviewer Jane Smith
1 - 5 of 7			
<input type="checkbox"/> Show User Comments <input type="checkbox"/> Show System Comments			
Refresh Close			

Figure 7-6 Activities in User Comments Using Descriptive Names

Process Comments			
Date	Activity	User	Comments
01/04/2007 05:04:52 PM	Manager Approval	IDMProv	User task assigned to reviewer Margo MacKenzie
01/04/2007 05:05:17 PM	Manager Approval	IDMProv	User task claimed by reviewer Margo MacKenzie
01/04/2007 05:05:20 PM	Manager Approval	IDMProv	User task approved by reviewer Margo MacKenzie
01/04/2007 05:05:21 PM	HR Approval	IDMProv	User task assigned to reviewer Human Resources
01/04/2007 05:05:50 PM	HR Approval	IDMProv	User task claimed by reviewer Jane Smith
1 - 5 of 7			
<input type="checkbox"/> Show User Comments <input type="checkbox"/> Show System Comments			
Refresh Close			

7.2.2 Defining the Data Item Mappings

You use the Data Item Mapping view to map data from the data flow into fields in a form (pre-activity mapping) and to map data from the form back to the data flow (post-activity mapping).

- 1 Right-click the activity icon for which you want to set properties and select *Show Data Item Mapping* from the menu.

TIP: You can also display the *Data Item Mappings* tab by selecting *Show Data Item Mapping* from the *PRD* menu.

The Data Item Mapping view is displayed:

Overview Workflow Forms		
Properties	Data Item Mapping	Email Notification Data Flow Provisioning View Policy Set Project Checker
<input checked="" type="radio"/> Pre Activity <input type="radio"/> Post Activity		
Source Expression	Target Form Field	Data Type
approval.getName(locale)	title	string
	subheading	string
initiator	initiator	string
recipient	recipient	string
process.getTimestamp()	initiatedTime	date
flowdata.get("reason")	reason	string
	apwaComment	string

- 2 For pre-activity mapping, click in the *Source Expression* field for the item that you want to map, then specify an expression. For post-activity mapping, click in the *Target Expression* field for the item that you want to map, then specify an expression.

Pre-activity maps can be used for

- ♦ Initializing form control values.
- ♦ Setting default values for form controls.
- ♦ Populating complex form controls with data lists derived from LDAP Queries.
- ♦ Passing data from form controls of a previous activity to a form control in the current activity.
- ♦ Calling external Java* classes to process data.

Post-activity maps can be used for

- ♦ Creating new data items in flowdata.
- ♦ Moving form control data from an activity into flowdata.
- ♦ Calling external Java* classes to process data.

For detailed information about data item mapping for the different types of activities, see [Chapter 8, “Workflow Activity Reference,” on page 179](#).

The Start Activity can have hard-coded strings, system variables like process locale and recipient, and Identity Vault expressions (created using the ECMA expression builder VDX Expr Panel) in pre-activity maps.

Leave the Source Expression blank in pre-activity maps for form fields that the user is expected to fill in. Alternatively, create a source expression to supply a default value for form fields that the user is expected to fill in. In either case the form field needs to be defined as editable. See [Section 6.5.2, “General Form Control Properties,” on page 124](#) for information about setting the properties of form fields.

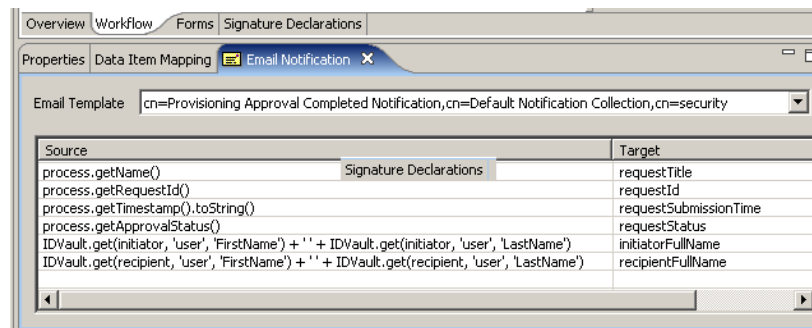
7.2.3 Defining the Email Notification Settings

You use the *Email Notification* view to select an e-mail template, and to specify expressions to provide values for named parameters included in the e-mail template. E-mails are sent when a new Approval activity starts (to notify the approver that they have work to do) and when the Finish activity completes (to notify the initiator that the workflow is done).

- 1 Right-click the activity icon for which you want to set properties and select *Show Email Notification* from the menu.

TIP: You can also display the *Email Notification* tab by selecting *Show Email Notification* from the *PRD* menu.

The *Email Notification* view is displayed:



- 2 Click the *Email Template* field, then select an e-mail template from the list of defined templates.

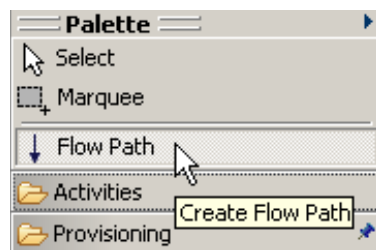
Editing an e-mail template: You can edit an e-mail template in Designer. To do this, select an Identity Vault in the *Modeler*, then scroll to *Default Notification Collection* in the *Outline View*. Right-click a template, then select *Edit Template*.

- 3 Click in the *Source* field for a *Target* token and specify an ECMAScript expression that assigns a value to the token.

See [Chapter 8, “Workflow Activity Reference,” on page 179](#) for information about e-mail notification settings.

7.3 Adding the Flow Paths

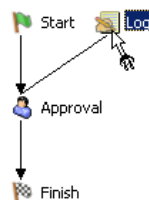
- 1 Click the Flow Path tool in the palette:



The mouse pointer turns into a flow path pointer:



- 2 Click the activity from which you want the flow path to begin, then click the activity on which you want the flow path to end:



The activities are connected.

- 3 To configure the flow path, click the Select tool in the palette, right-click the flow path, then select *Show Properties*.

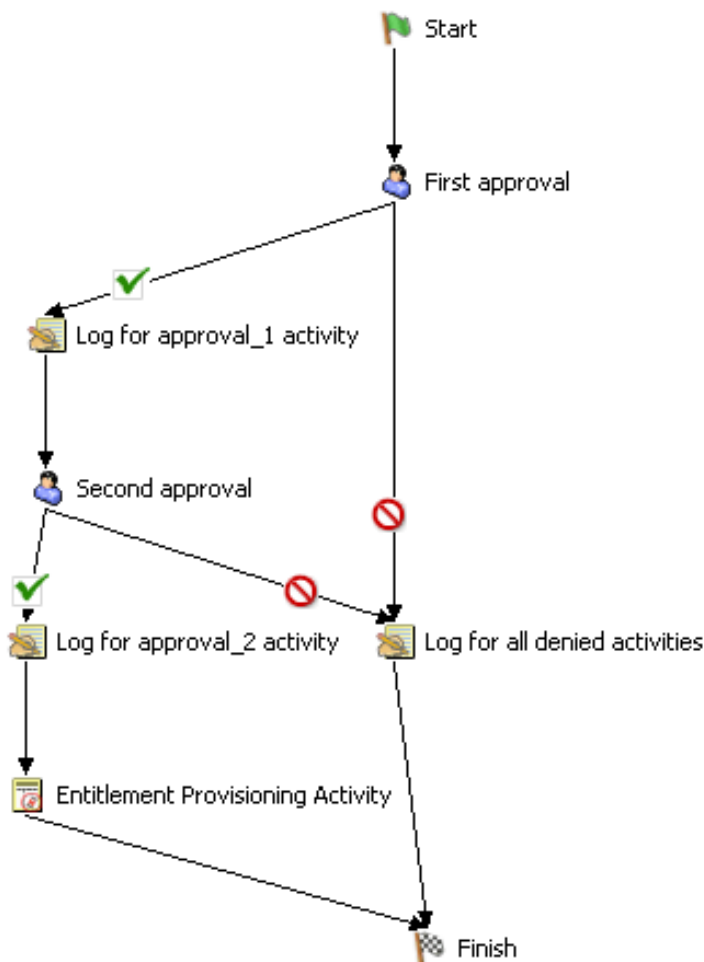
For information about configuring flow paths, see [Section 7.4, “Configuring Flow Paths,”](#) on [page 165](#).

7.4 Configuring Flow Paths

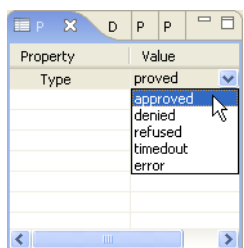
After you have added a flow path to a workflow diagram, you can specify the path type. For details on adding flow paths to a workflow, see [Section 7.3, “Adding the Flow Paths,”](#) on [page 164](#).

To configure a flow path:

- 1 Click the flow path in the workflow diagram.



- Set the flow type on the *Properties* tab by selecting one of the options in the *Type* drop-down list.



The flow path types are described in the following table:

Flow Type	Description
forward	<p>Forwards control to the next activity in a workflow.</p> <p>The forward flow path is available after all activities except:</p> <ul style="list-style-type: none"> ♦ Approval ♦ Condition ♦ Finish
approved	<p>Determines what happens when a user approves a request.</p> <p>The approved flow path is valid only after the Approval activity.</p>
denied	<p>Determines what happens when a user denies a request.</p> <p>The denied flow path is valid only after the Approval activity.</p>
refused	<p>Determines what happens when a user refuses a request.</p> <p>The refused flow path is valid only after the Approval activity.</p>
timeout	<p>Determines what happens when an Approval activity times out because the user did not respond.</p> <p>The timeout flow path is valid only after the Approval activity.</p>
error	<p>Determines what happens when an Approval or Condition activity terminates with an error.</p> <p>The error flow path is valid only after the Approval and Condition activities.</p>
true	<p>Determines what happens when a conditional expression evaluates to true.</p> <p>The true flow path is valid only after the Condition activity.</p>
false	<p>Determines what happens when a conditional expression evaluates to false.</p> <p>The false flow path is valid only after the Condition activity.</p>

If the *Properties* tab is not displayed, right-click the flow path in the workflow diagram and select *Show Properties*.

7.5 Addressing an Approval Activity

To address an Approval activity, you must enter a valid expression for the *Addressee* property. In addition, the final number of approvals that are required to approve the activity is determined by the relationship between the *Addressee* property and the *Approver Type* property.

NOTE: If the expression specified in the *Addressee* property of an Approval activity evaluates to a non-existent DN (for example, if the expression was hard-coded incorrectly, calculated incorrectly, or submitted incorrectly by a user selection), no indication is given that the workflow is not processing normally, when it is in fact orphaned. The application server console displays a normal forward message, and the Comment and Flow history shows a normal “assigned” message. To avoid this problem, we recommend that you follow these best practices:

1. Use a Condition activity before the Approval activity and validate the addressee in the Condition activity.
 2. Since the addressee could still be deleted after the addressee is validated in the Condition activity, you should specify, for the Approval activity, a timeout interval and a link that performs the desired action in case the workflow times out.
-

7.5.1 Valid Addressee Expressions

An Addressee expression (including expressions that return data abstraction layer Entities) must resolve to one of the following at runtime:

- ♦ A valid individual addressee that can be a user DN or a group DN.
- ♦ A valid list of addressees (for example, created using a Java vector object) that can contain multiple User DNs, or multiple group DNs, or a mixture of both.

The maximum number of approvals possible equals the number of Addressees (the number of User DNs plus the number of Group DNs) and does not include or count the individual members of a Group.

NOTE: A Group DN is always processed to contribute a single vote (that is, when one member of a group claims an activity, the rest of the members of the group can no longer see or claim the activity), regardless of the *Approver Type*.

The following table provides examples of valid addressee expressions that you can create using the ECMA expression builder.

Table 7-4 Examples of Addressee Expressions

Type of Expression	Example
Individual user DN	'cn=jdoe,ou=users,ou=mysample,o=myorg'
Individual group DN	'cn=Accounting,ou=groups,ou=mysample,o=myorg'

Type of Expression	Example
A vector of DNs (can include user or group DNs)	<pre>function DNVector() { v=new java.util.Vector(); v.add('CN=jdoe,' + USER_CONTAINER); v.add('CN=Accounting,' + GROUP_CONTAINER); v.add('CN=jsmith,' + USER_CONTAINER); v.add('CN=bsmith,' + USER_CONTAINER); return v; }; DNVector();</pre> <p>In this example, the total number of addressees is four (three individuals and one user from the Accounting group).</p>

7.5.2 Relationship Between Addressee and Approver Type

The behavior of the workflow and the total number of affirmative approvals needed varies depending on the type of Addressee that is specified by the Addressee expression, and the Approver Type that is selected.

Normal Approver Type

The following table describes the workflow behavior when different types of addressee are used with the Normal Approver Type.

Table 7-5 Workflow Behavior with Normal Approver Type

Addressee Value	Description
Individual User DN or Entity	<ul style="list-style-type: none"> Only the user can see the <i>Approval</i> activity in their Task List. Only one approval is needed to complete the activity as Approved.
Individual Group DN or Group Entity	<ul style="list-style-type: none"> Each member of Group can see the activity in task list. When one member claims the activity, it is removed from the task lists of others. Only one approval is needed to complete the activity as Approved.
Multiple User DNs or User Entities (Virtual Group of Users)	Not allowed.
Multiple Group DNs or Group Entities (Virtual Group of Groups)	Not allowed.
Mixture of Users and Groups (Virtual Group Mixture)	Not allowed.

Group Approver Type

The following table describes the workflow behavior when different types of addressee are used with the Group Approver Type.

Table 7-6 *Workflow Behavior with Group Approver Type*

Addressee Value	Description
Individual User DN or Entity	<ul style="list-style-type: none"> ♦ Only the user can see the Approval activity in their task list. ♦ Only one approval is needed to complete the activity as Approved.
Individual Group DN or Group Entity	<ul style="list-style-type: none"> ♦ Each member of Group can see the the activity in their task list. ♦ When one member claims the activity, it is removed from task lists of others. ♦ Only one approval is needed to complete the activity as Approved.
Multiple User DNs or User Entities (Virtual Group of Users)	<ul style="list-style-type: none"> ♦ Each user in the virtual group can see the activity in their task list. ♦ When one user from the virtual group claims the activity, the activity is removed from the task lists of others. ♦ Only one approval is needed to complete the activity as Approved.
Multiple Group DNs or Group Entities (Virtual Group of Groups)	<ul style="list-style-type: none"> ♦ Each member in each of the groups can see the activity in their task list. ♦ When one user from the virtual group claims the activity, the activity is removed from the task lists of others in all of the groups. ♦ Only one approval is needed to complete the activity as Approved.
Mixture of Users and Groups (Virtual Group Mixture)	<ul style="list-style-type: none"> ♦ Each user and member of each Group of the mixed virtual group can see the activity in their task list. ♦ When one member from the virtual group claims the activity, the activity is removed from the task lists of others. ♦ Only one approval is needed to complete the activity as Approved.

Multiple Approver Type

The following table describes the workflow behavior when different types of addressee are used with the Multiple Approver Type.

Table 7-7 *Workflow Behavior with Multiple Approver Type*

Addressee Value	Description
Individual User DN or Entity	<ul style="list-style-type: none"> ♦ Only the user can see the activity in their task list. ♦ Only one approval is needed to complete the activity as Approved.
Individual Group DN or Group Entity	<ul style="list-style-type: none"> ♦ Each member of the group can see the activity in their task list. ♦ When one member claims the activity, the activity is removed from the task lists of others. ♦ Only one approval is needed to complete the activity as Approved.
Multiple User DNs or User Entities (Virtual Group of Users)	<ul style="list-style-type: none"> ♦ Each user in the virtual group can see the activity in their task list. ♦ Each user can claim the activity. ♦ Approval of each user is needed to complete the activity as Approved. ♦ Any single denial completes the activity as Denied.

Addressee Value	Description
Multiple Group DNs or Group Entities (Virtual Group of Groups)	<ul style="list-style-type: none"> Each member in each of the groups can see the activity in their task list. When one member from a group claims the activity, the activity is removed from the task list of others in that Group. Each group must supply one approval to complete the activity as Approved. Any single denial completes the activity as Denied.
Mixture of Users and Groups (Virtual Group Mixture)	<ul style="list-style-type: none"> Each user and each member of each group of the mixed virtual group can see the activity in their task list. Each user can claim the activity, and one member of each group can claim the activity (others in group will then not see the task). Each user and one member of each group must approve to complete the activity as Approved. Any single denial completes the activity as Denied.

Quorum Approver Type

The following table describes the workflow behavior when different types of addressee are used with the *Quorum Approver Type*.

Table 7-8 Workflow Behavior with *Quorum Approver Type*

Addressee Value	Description
Individual User DN or Entity	<ul style="list-style-type: none"> Only the user can see the activity in their task list. Only one approval is needed to complete the activity as Approved.
Individual Group DN or Group Entity	<ul style="list-style-type: none"> Each member of the group can see the activity in their task list. When one member claims the activity, the activity is removed from the task lists of others. Only one approval is needed to complete the activity as Approved.
Multiple User DNs or User Entities (Virtual Group of Users)	<ul style="list-style-type: none"> Each user in the virtual group can see the activity in their task list. All users in the virtual group can claim the activity simultaneously. An absolute number or specified percentage of Addressees must approve to complete the activity as Approved.
Multiple Group DNs or Group Entities (Virtual Group of Groups)	<ul style="list-style-type: none"> Each member in each group can see the activity in their task list. One member of each group can claim the task (others in group will then not see the task). An absolute number or specified percentage of Addressees must approve to complete the activity as Approved.

Addressee Value	Description
Mixture of Users and Groups (Virtual Group Mixture)	<ul style="list-style-type: none"> ♦ Each user and each member of each Group of the mixed virtual group can see the activity in their task list. ♦ Each user can claim the activity, and one member of each group can claim the activity (others in group will then not see task). ♦ An absolute number or specified percentage of Addressees must approve to complete the activity as Approved.

7.6 Provisioning Multiple Individuals with One Workflow Instance

You can configure a provisioning request definition so that one individual (for example, a team manager) can provision multiple individuals (for example, members of a team, or a group) with one workflow. The provisioning request definition can be configured to provision any one of the following:

- ♦ multiple individual users from the default user container
- ♦ all members of a group from the default group container (for example, Sales, Marketing, HR, IT)
- ♦ all members of any arbitrary Identity Vault container

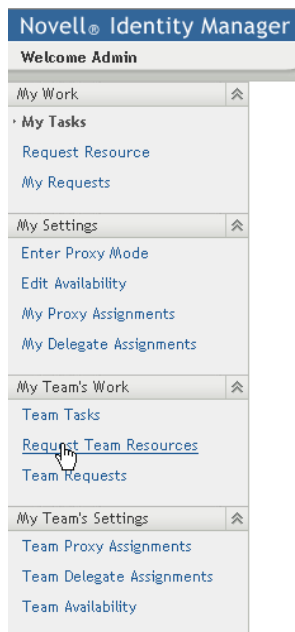
To create this type of workflow, create the provisioning request definition as you normally would. On the Overview tab, select *Single Flow Provision Members* from the *Flow Strategy* list.

7.6.1 Basic Steps for using the Workflow

This section describes the basic steps for using a workflow that utilizes the *Single Flow Provision Members* flow strategy.

- 1 Log in to the user application as a user application administrator.
- 2 Click *Requests and Approvals*.

3 Click *Request Team Resources*.



4 Select the provisioning category to which the provisioning request belongs, then click *OK*.

A screenshot of the 'Request Team Resources' dialog box, Step 1 of 4. The text says 'Step 1 of 4: Select the category of the resource you are requesting.' Below this is a 'Resource Search Criteria' dropdown menu with 'All' selected. A 'Continue' button is at the bottom left. The dropdown menu is open, showing options: 'Accounts', 'All', 'Entitlements', and 'Groups'.

You should see a workflow that is marked with an icon that contains a cluster of people:

A screenshot of the 'Request Team Resources' dialog box, Step 2 of 4. The text says 'Step 2 of 4: Select the resource from the list.' Below this is a table with three columns: 'Resource', 'Resource Search Criteria', and 'Description'. The table contains one row with a cluster of people icon, the resource name 'Strategy_SingleFlowProvisionMembers', the search criteria 'Accounts', and the description 'Strategy_SingleFlowProvisionMembers'. Below the table is a 'Back' button. The text '1 - 1 of 1' is centered below the table.

5 Click the name of the workflow.

A form is displayed that provides three methods of 3 methods of selecting multiple users to provision:

- ♦ Specify one or more recipients
- ♦ Specify a group
- ♦ Specify a container

6 Specify the recipients, then click *Continue*.

7.6.2 Setting up the Workflow for a Team Manager to Use

To enable a Team Manager to use a work flow that uses the *Single Flow Provision Members* flow strategy, you need to perform these additional setup steps.

- 1 Login to iManager as an administrator.
- 2 In *Roles and Tasks*, select *Provisioning Configuration*.
- 3 Select *Provisioning Teams*.
- 4 Setup the team if it is not already setup.
- 5 Bind the workflow to the team by defining a *Provisioning Team Request* using the *Provisioning Configuration* Role and Task.

For information about defining a provisioning team request, see “[Managing Provisioning Team Request Rights](#)” in the *Identity Manager 3.5 User Application: Administration Guide*.

7.7 Working with Entity Activities

You use Entity activities to update entities in the Identity Vault. The procedures for working with Entity activities differ slightly from the procedures for working with other activity types.

The section includes the following topics:

- ♦ [Section 7.7.1, “Adding or Modifying an Entity,” on page 173](#)
- ♦ [Section 7.7.2, “Using an Entity Activity to Delete an Entity,” on page 174](#)
- ♦ [Section 7.7.3, “Using an Entity Activity to Delete an Attribute or Value,” on page 174](#)

7.7.1 Adding or Modifying an Entity

- 1 From the Workflow page, click the Entity activity icon in the palette, then click the canvas to insert the Entity activity into the workflow.
- 2 Click the *Properties* tab.

- 3 Click in the *Value* column of the *Entity Type* field, then select the *Entity Type* (for example, User, Group) that you want to create or modify. If the target object that you specify in [Step 6](#) already exists, the target object is modified; if the target object doesn't exist, it is created.
 - 4 Click in the *Value* column of the *Operation* field, then select *Create/Modify*.
 - 5 Click the Data Item Mapping tab.
 - 6 Click the button next to the *Entity dn* field to display the ECMA expression builder, then specify an expression that identifies the target of the operation (for example, “recipient”).
 - 7 Click *OK* to return to the Data Item Mapping view.
 - 8 Specify expressions for other attributes as required to create the Entity.
- See [Section 3.2, “Working with Entities and Attributes,” on page 41](#) for information about adding entities. If you are adding an entity, you must enter expressions for all required attributes.

7.7.2 Using an Entity Activity to Delete an Entity

- 1 From the Workflow page, click the Entity activity icon in the palette, then click the canvas to insert the Entity Activity into the workflow.
- 2 Click the *Properties* tab.
- 3 Click in the *Value* column of the *Entity Type* field, then select the Entity Type (for example, User, Group) to which the entity that you want to delete belongs.
- 4 Click in the *Value* column of the *Operation* field, then select *Delete entity*.
- 5 Click the *Data Item Mapping* tab.
- 6 Click the button next to the *Entity dn* field to display the ECMA expression builder, then specify an expression that identifies the Entity that you want to delete.
- 7 Click *OK* to return to the Data Item Mapping view.

7.7.3 Using an Entity Activity to Delete an Attribute or Value

- 1 From the Workflow page, click the Entity activity icon in the palette, then click the canvas to insert the Entity Activity into the workflow.
- 2 Click the *Properties* tab.
- 3 Click in the *Value* column of the *Entity Type* field, and select the Entity Type (for example, User, Group) of the entity to which the attribute or value that you want to delete belongs.
- 4 Click in the *Value* column of the *Operation* field, and select *Delete attribute/value*.
- 5 Click the Data Item Mapping tab.
- 6 Click the button next to the *Entity dn* field to display the ECMA expression builder, then specify an expression that identifies the entity that contains the attribute or value that you want to delete.
- 7 Click *OK* to return to the Data Item Mapping view.
- 8 Click in the *Delete Type* field for the attribute to which you want the operation to apply, then select the operation from the list:
 - ♦ Select *Delete Attribute* for single-value attributes

- ♦ Select either *Delete Attribute* or *Delete Value* for multi-value attributes. Selecting *Delete Value* for multi-value attributes also requires that you to enter an expression to identify the value that you want to delete.
- 9 To delete a value, click in the *Delete Value Expression* field for the attribute to which you want the operation to apply, then specify an expression that resolves to the value of the attribute that you want to delete.

7.8 Configuring Digital Signature Support

This section describes how to use Designer to configure provisioning request definitions to support digital signatures. To configure a provisioning request definition to support digital signatures, follow the steps outlined in the following table.

Table 7-9 Steps for Specifying Digital Signature Support in Workflows

Step	Task	Description
1	Create one or more digital signature declarations.	See Section 7.8.2, "Creating a Signature Declaration," on page 176.
2	Specify whether a digital signature is required to initiate a provisioning request.	In the Workflow tab, click the Start Activity and set the following properties: <ul style="list-style-type: none"> ♦ <i>Digital Signature Required:</i> See Section 7.8.1, "Digital Signature Workflow Properties," on page 176. ♦ <i>Signature Declaration:</i> Choose a signature declaration from the dropdown. The list is only populated if you completed Step 1 (above).
3	Specify whether a digital signature is required for each approval step within the workflow.	Each approval step can have more than one outgoing link. You must specify the Digital Signature Required property and the Signature Declaration properties for each approval step and each outgoing flow path. For a description of the property settings, see Section 7.8.1, "Digital Signature Workflow Properties," on page 176.
4	Determine for the request and each approval form whether it contains a title control.	Title controls have a property called <i>Display title in signed form document</i> . Determine for your application and use of digital signatures whether this property should be set to true or false. For more information on this property, see Section 6.5.19, "Title," on page 147.

7.8.1 Digital Signature Workflow Properties

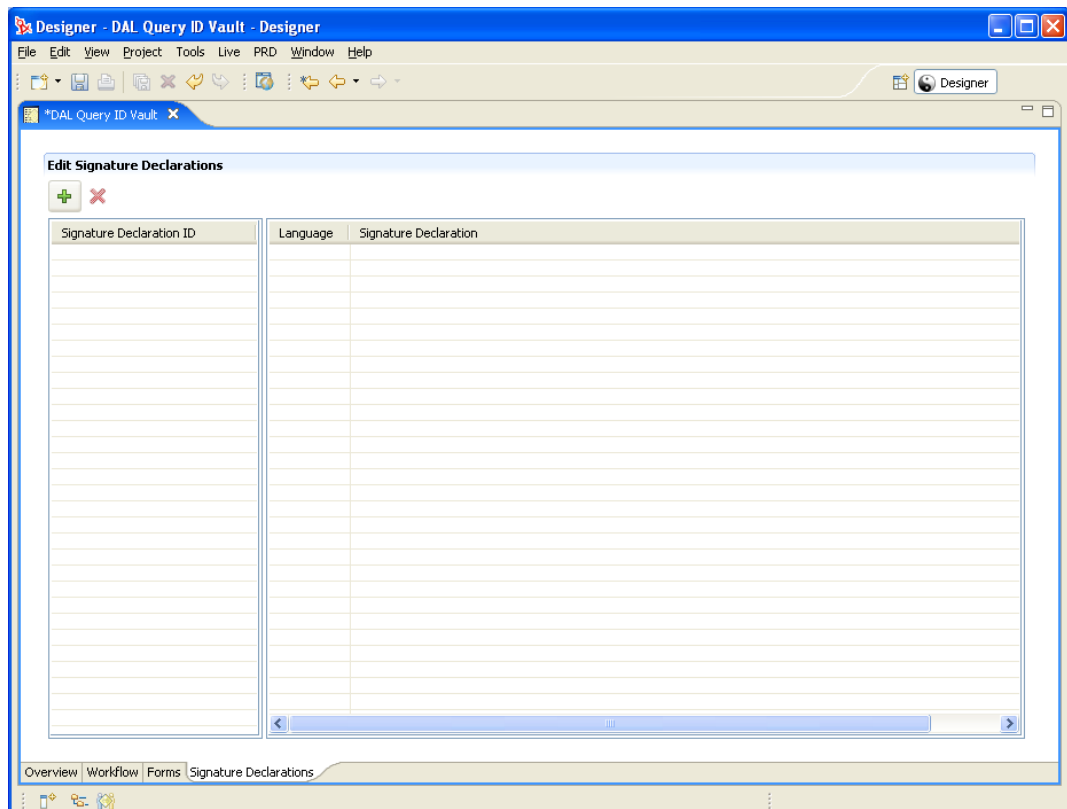
Table 7-10 *Digital Signature Settings*

Setting	Description
Digital Signature Type	<p>Specifies whether the digital signature will use data or form as its type:</p> <ul style="list-style-type: none">♦ Data specifies that the XML signature serves as the user agreement. When Data is selected, the XML data is written to the audit log.♦ Form specifies that a PDF document that includes the digital signature declaration be generated. This document serves as the user agreement. The user can preview the generated PDF document before submitting a request or approval. When Form is selected, the PDF document (encapsulated in XML) is written to the audit log.
Digital Signature Declaration	<p>Specifies a digital signature confirmation string that certifies the user's signature. See Section 7.8.2, "Creating a Signature Declaration," on page 176.</p>

7.8.2 Creating a Signature Declaration

To create a signature declaration:

- 1 Open the *Signature Declarations* tab.



2 Click to add a row, then fill in the fields as follows:

Field	Description
Signature Declaration ID	A unique identifier for the signature declaration. This ID is displayed in the drop-down for the "Digital Signature Declaration" on page 176.
Language	Choose a language and specify the signature declaration translation for that language. The signature declaration string is also exported as part of the Provisioning view's <i>Export > Export Localization Data</i> so that you can send the declaration to be localized as part of the rest of the User Application display labels and strings.
Signature Declaration	The string to display in a form as the signature declaration.

3 Click *Save*.

Workflow Activity Reference

8

This section provides details on configuring the different types of workflow activities. The display names for all activities can be localized by clicking the localization button (see [Section 2.9, “Localizing Display Labels,” on page 31](#)) for the activity name property. Activity display names are also exported as part of the Provisioning view’s *Export > Export Localization Data* (see [Section 2.9.3, “Exporting and Importing Localized Labels,” on page 33](#)) so that you can send the activity names to be localized as part of the rest of the User Application display labels and strings.

Topics in this section include:

- ♦ [Section 8.1, “Start Activity,” on page 179](#)
- ♦ [Section 8.2, “Approval Activity,” on page 181](#)
- ♦ [Section 8.3, “Log Activity,” on page 188](#)
- ♦ [Section 8.4, “Branch Activity,” on page 189](#)
- ♦ [Section 8.5, “Mapping Activity,” on page 189](#)
- ♦ [Section 8.6, “Merge Activity,” on page 190](#)
- ♦ [Section 8.7, “Condition Activity,” on page 191](#)
- ♦ [Section 8.8, “Workflow Status,” on page 192](#)
- ♦ [Section 8.9, “Finish Activity,” on page 193](#)
- ♦ [Section 8.10, “Integration Activity,” on page 194](#)
- ♦ [Section 8.11, “Entitlement Activity,” on page 196](#)
- ♦ [Section 8.12, “Entity Activity,” on page 198](#)

8.1 Start Activity

The Start activity is the first activity to execute in a workflow. It begins execution when the user makes a request to provision a resource. After the user makes the request, the Start activity displays the initial request form to the user. On the initial request form, the user can be asked to specify a comment that indicates the reason for the request.

You can customize the initial request form to suit your application requirements. For details on customizing forms, see [Chapter 6, “Creating Forms for a Provisioning Request Definition,” on page 103](#).

Before displaying the form to the user, the Start activity performs any pre-activity data mappings specified for the activity.

After the user submits the form, the Start activity performs any post-activity data mappings specified for the activity. These mappings typically include copying data from form fields into the flowdata object.

8.1.1 Properties

The Start activity has the following property:

Table 8-1 *Start Activity Property*

Property Name	Description
Name	Provides a name for the activity.
Digital Signature Type	See Digital Signature Type in Table 7-10 on page 176 .

8.1.2 Data Item Mapping

To bind the data items associated with the Start activity, you define pre-activity and post-activity mappings. The pre-activity mappings initialize data in the request form with constants or values retrieved from the flowdata object. The post-activity mappings move form data back into the flowdata object.

Table 8-2 *Start Activity Data Item Mappings*

Setting	Description
Pre-Activity	<p>Allows you to specify one or more pre-activity mappings. When this option is selected, you can double-click a cell in the <i>Source Expression</i> column to specify where the initial request form gets data for a particular target form field.</p> <hr/> <p>NOTE: When the <i>Pre-Activity</i> option is selected, the cells in the <i>Target Form Field</i> column are not editable.</p>
Post-Activity	<p>Allows you to specify one or more post-activity mappings. When this radio button is selected, you can double-click a cell in the <i>Target Expression</i> column to specify where data from a form field should be copied after the form has been processed.</p> <hr/> <p>NOTE: When the <i>Post-Activity</i> option is selected, the cells in the <i>Source Form Field</i> column are not editable.</p>
Source Expression	Specifies a source expression for a pre-activity mapping. When you click a cell in the <i>Source Expression</i> column, the ECMA expression builder displays to help you define your expression.
Target Expression	Specifies a target expression for a post-activity mapping. When you click a cell in the <i>Target Expression</i> column, the ECMA expression builder displays to help you define your expression.

For details on building ECMA expressions, see [Chapter 10, “Working with ECMA Expressions,” on page 249](#).

8.1.3 Email Notification

Not supported with this activity.

8.2 Approval Activity

The Approval activity is a user-facing activity that displays an approval form to the user. On the approval form, the user can approve, deny, or refuse a provisioning request. The Approval activity can have multiple outgoing flow paths, but only one of the paths is executed at runtime.

You can customize the approval form to suit your application requirements. For details on customizing forms, see [Chapter 6, “Creating Forms for a Provisioning Request Definition,” on page 103](#).

Before displaying the form to the user, the Approval activity performs any pre-activity data mappings specified for the activity.

After the user submits the form, the Approval activity performs any post-activity mappings specified for the activity. These mappings typically include copying data from form fields into the flowdata object.

8.2.1 Properties

The Approval activity has the following properties:

Table 8-3 *Approval Activity Properties*

Property Name	Description
<i>Name</i>	Provides a name for the activity.
<i>Addressee</i>	<p>Specifies a dynamic expression that identifies the addressee for the activity. The addressee is determined at runtime, based on how the expression is evaluated.</p> <p>For more information about developing valid Addressee expressions, and about how Addressee interacts with the Approver Type property, see Section 7.5, “Addressing an Approval Activity,” on page 167.</p> <hr/> <p>TIP: To simplify the process of testing a new workflow, you can set the addressee to be the recipient. This removes the need to log out of the User Application and log in again as a manager each time you want to test your forms. This technique is particularly useful when the workflow involves multiple levels of approval. After the testing phase is complete, you can change the addressee to the correct value.</p> <hr/> <p>For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 249. For descriptions of the system variables available in a workflow, see Section 4.3.3, “Understanding Workflow Data,” on page 86.</p>

Property Name	Description
<i>Reminder Start</i>	<p>Specifies a dynamic expression that defines, in milliseconds, the time at which the first reminder e-mail should be sent. The start value is an offset from the time of the first assignment associated with the activity. You can pick pre-defined expressions that represent common intervals (for example, hour, day, week) in the <i>ECMAScript Variables</i> pane of the ECMA expression builder.</p> <p>This is part of the reminder e-mail function. If this activity is considered important and needs to be acted on quickly, you can configure the activity to send a reminder e-mail to the activity addressee. For example, you can set the reminder settings to send a reminder e-mail 5 days before the activity times out, and on a daily basis until the activity times out. To do this, specify a <i>Reminder Start</i> time, a <i>Reminder Interval</i>, and the e-mail to be sent (see Section 8.2.3, “E-mail Notification,” on page 186).</p> <p>For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 249. For descriptions of the system variables available in a workflow, see Section 4.3.3, “Understanding Workflow Data,” on page 86.</p>
<i>Reminder Interval</i>	<p>Specifies a dynamic expression that defines the interval between which reminder e-mails are sent. You can pick pre-defined expressions that represent common intervals (for example, hour, day, week) in the <i>ECMAScript Variables</i> pane of the ECMA expression builder.</p>
<i>Escalation Addressee</i>	<p>Specifies a dynamic expression that identifies the user who should get this task if the timeout limit has been reached.</p> <p>The escalation addressee is determined at runtime, based on how the expression is evaluated.</p> <p>For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 249. For descriptions of the system variables available in a workflow, see Section 4.3.3, “Understanding Workflow Data,” on page 86.</p>

Property Name	Description
<i>Escalation Count</i>	<p>Specifies the number of times to retry the activity in the event of a timeout.</p> <p>When an activity times out, the workflow process can try to complete the activity again, depending on the escalation count specified for the activity. With each retry, the workflow process can escalate the activity to another user. In this case, the activity is reassigned to another user (the user's manager, for example) to give this user an opportunity to finish the work of the activity. If the last retry times out, the activity can be marked as approved, denied, refused, timedout, or in error, depending on the final timeout action specified for the activity.</p> <p>The <i>Timeout</i> interval (see <i>Timeout</i> in this table) takes precedence over the <i>Escalation Interval</i>. For example, if you set the timeout to 10 minutes, and specify an Escalation Count of 3 and Escalation Interval of 5 minutes, the activity will finish after 10 minutes without attempting all of the retries. In this example, the second retry would be canceled, and the workflow would finish processing for the activity. At the conclusion of the activity, the workflow engine would follow the link defined by the final timeout action.</p>
<i>Escalation Interval</i>	<p>Specifies a dynamic expression that defines the period of time allotted for the addressee to complete the task. The escalation interval applies each time the activity is executed by the addressee.</p> <p>The <i>Timeout</i> interval (see <i>Timeout</i> in this table) takes precedence over the <i>Escalation Interval</i>. For example, if you set the timeout to 10 minutes, and specify an Escalation Count of 3 and Escalation Interval of 5 minutes, the activity will finish after 10 minutes without attempting all of the retries. In this example, the second retry would be canceled, and the workflow would finish processing for the activity. At the conclusion of the activity, the workflow engine would follow the link defined by the final timeout action.</p> <p>For details on building ECMA expressions, see Chapter 10, "Working with ECMA Expressions," on page 249. For descriptions of the system variables available in a workflow, see Section 4.3.3, "Understanding Workflow Data," on page 86.</p>
<i>Escalation Reminder Start</i>	<p>Specifies a dynamic expression that defines the time at which the first reminder e-mail (see <i>Reminder Start</i> in this table) should be sent to the <i>Escalation Addressee</i>. The start value is an offset from the time of the escalation assignment. You can pick pre-defined expressions that represent common intervals (for example, hour, day, week) in the <i>ECMAScript Variables</i> pane of the ECMA expression builder.</p>
<i>Escalation Reminder Interval</i>	<p>Specifies a dynamic expression that defines how often messages are sent to the <i>Escalation Addressee</i> after the first escalation reminder is sent. You can pick pre-defined expressions that represent common intervals (for example, hour, day, week) in the <i>ECMAScript Variables</i> pane of the ECMA expression builder.</p>

Property Name	Description
<i>Final Timeout Action</i>	<p>Determines the final state of the request in the event that the workflow times out. The choices are</p> <ul style="list-style-type: none"> ♦ approved ♦ denied ♦ refused ♦ timedout ♦ error
<i>Timeout</i>	<p>Specifies a dynamic expression that defines the period of time allotted for the addressee to complete the task. The timeout interval applies each time the activity is executed by the addressee.</p> <p>The Timeout setting takes precedence over the <i>Escalation Count</i> and <i>Escalation Interval</i> values. If the Timeout setting for the activity is reached before one or more of the escalation attempts have been tried, the activity finishes processing without executing these escalation attempts. For example, if you set the timeout to 10 minutes, and specify an Escalation Count of 3 and Escalation Interval of 5 minutes, the activity will finish after 10 minutes without attempting all of the escalation attempts. In this example, the second escalation attempt would be canceled, and the workflow would finish processing for the activity. At the conclusion of the activity, the workflow engine would follow the link defined by the final timeout action.</p> <p>For details on building ECMA expressions, see Chapter 10, "Working with ECMA Expressions," on page 249. For descriptions of the system variables available in a workflow, see Section 4.3.3, "Understanding Workflow Data," on page 86.</p>
<i>Time Units</i>	<p>Determines the unit of measure used for the timeout interval. The choices are</p> <ul style="list-style-type: none"> ♦ Milliseconds ♦ Days ♦ Hours ♦ Minutes ♦ Seconds
<i>Form</i>	<p>Specifies the name of the approval form to display to the user.</p> <p>An Approval activity must have a form associated with it. If no form is specified, an error message is displayed at runtime.</p>
<i>Exclude Requestor</i>	<p>Specifies whether a requestor can approve their own provisioning requests. Select true to allow a requestor to approve their own provisioning requests; otherwise, select false.</p>

Property Name	Description
<i>Approver Type</i>	<p>Specifies the number of addresses that are allowed and the approval pattern that is enforced for this activity. The choices are</p> <ul style="list-style-type: none"> ♦ Normal - Action by the addressee is required to complete the approval. ♦ Group - Action by one addressee in the group is required to complete the approval. ♦ Multiple - Action by all of the addressees is required to complete the approval. <p>You cannot use post activity data item mapping with the <i>Multiple Approver Type</i>.</p> <ul style="list-style-type: none"> ♦ Quorum - Action by a percentage of addressees or an absolute number of addressees (see <i>Quorum</i> property in this table) is required to complete the approval. <p>You cannot use post activity data item mapping with the <i>Quorum Approver Type</i>.</p> <p>For information about how the <i>Approver Type</i> property interacts with the <i>Addressee</i> property, see Section 7.5, “Addressing an Approval Activity,” on page 167.</p>
<i>Notify by Email</i>	<p>Specifies whether this activity should send e-mail notifications. Set to true to notify by e-mail; otherwise, set to false.</p> <p>You specify the e-mail to send using the E-Mail Notification tab (see Section 8.2.3, “E-mail Notification,” on page 186).</p> <p>To use this feature, the <i>Notify participants by Email</i> parameter for the provisioning request definition must be set to true (see Table 5-1, “Overview Properties,” on page 101).</p>
<i>Quorum</i>	<p>Creates an expression that specifies a percentage (for example, '75%') of approvals that is required before a quorum is achieved, or an absolute number (for example, '3') of approvals that are required before a quorum is achieved.</p>
<i>Digital Signature Type</i>	<p>See Digital Signature Type in Table 7-10 on page 176.</p>
<i>Priority</i>	<p>Specifies a dynamic expression that defines the priority of the approval activity. Valid priority values are 1, 2, or 3. You can also define an expression to determine the priority from workflow data. For example, <code>flowdata.get("Priority")</code>.</p> <p>In the User Application, the user can sort their list of tasks by the priority values of the tasks.</p>

8.2.2 Data Item Mapping

To bind the data items associated with the Approval activity, you define pre-activity and post-activity mappings. The pre-activity mappings initialize data in the approval form with constants, values retrieved from the flowdata object, system process variables, system activity variables, and data retrieved via expression calls to the directory abstraction layer. The post-activity mappings move form data back into the flowdata object.

Table 8-4 Approval Activity Data Item Mappings

Setting	Description
<i>Pre Activity</i>	<p>Allows you to specify one or more pre-activity mappings. When this option is selected, you can double-click a cell in the <i>Source Expression</i> column to specify where the approval form gets data for a particular target form field.</p> <hr/> <p>NOTE: When the <i>Pre-Activity</i> choice is selected, the cells in the <i>Target Form Field</i> column are not editable.</p> <hr/>
<i>Post Activity</i>	<p>Allows you to specify one or more <i>Post Activity</i> mappings. When this option is selected, you can double-click a cell in the <i>Target Expression</i> column to specify where data from a form field should be copied after the form has been processed.</p> <p>You cannot use <i>Post Activity</i> mapping with the <i>Multiple</i> and <i>Quorum</i> approver types (see Section 8.2.1, “Properties,” on page 181).</p> <p>The form for an Approval activity includes a special internal control called <code>apwaComment</code>. This control causes user comments to be written to the workflow database. It should not have a post-activity mapping. For more information on this control, see Section 6.5.9, “DNMaker,” on page 133.</p> <hr/> <p>NOTE: When the <i>Post-Activity</i> option is selected, the cells in the <i>Source Form Field</i> column are not editable.</p> <hr/>
<i>Source Expression</i>	<p>Specifies a source expression for a pre-activity mapping. When you click a cell in the <i>Source Expression</i> column, the ECMA expression builder displays to help you define your expression.</p>
<i>Target Expression</i>	<p>Specifies a target expression for a post-activity mapping. When you click a cell in the <i>Target Expression</i> column, the ECMA expression builder displays to help you define your expression.</p>

For details on building ECMA expressions, see [Chapter 10, “Working with ECMA Expressions,” on page 249](#).

8.2.3 E-mail Notification

To enable e-mail notification for the Approval activity, you need to specify the e-mail template to use, as well as source expressions for target tokens in the e-mail body.

Table 8-5 E-mail Notification Settings for the Approval Activity

Setting	Description
<i>Notify</i>	Specifies that this e-mail notification is a notification e-mail.
<i>Reminder</i>	Specifies that this e-mail notification is a reminder e-mail.
<i>Retry Reminder</i>	Specifies that this e-mail notification is a retry reminder e-mail.

Setting	Description
<i>Show System Tokens</i>	Displays system tokens (for example, TO, CC, BCC) in the <i>Target</i> column.
<i>Email Template</i>	<p>Specifies the name of the e-mail template to use. By default, the Approval activity uses the Provisioning Notification template.</p> <p>You can edit an e-mail template in Designer. For more information, see “Editing an e-mail template:” on page 164.</p>
<i>Source/Target</i>	<p>Specifies the source expressions for target tokens in the e-mail body.</p> <p>The list of target tokens is determined by the selected e-mail template. You cannot add new tokens, but you can assign values to the tokens by building your own source expressions. At runtime, source expressions are evaluated to determine the value of each token.</p> <p>The available target tokens are listed below:</p> <ul style="list-style-type: none"> ♦ TO ♦ CC ♦ BCC ♦ recipientFullName ♦ initiatorFullName ♦ requestTitle ♦ userFirstName <p>If you use a provisioning request definition template to create your workflow, each token has a default source expression. The default expressions retrieve values from the workflow process (the process object) or from the data abstraction layer (IDVault object). You can modify these expressions to suit your application requirements.</p> <hr/> <p>NOTE: When you create a workflow for use with the Resource Request portlet (see “Resource Request Portlet” in the <i>Identity Manager User Application: Administration Guide</i>) and you use <code>_default_</code> as the expression for the TO token as <code>_default_</code>, the addressee expression must be an IDVault expression.</p> <hr/> <p>For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 249.</p>
<p>NOTE: E-mail notification is only supported when the <i>Notify participants by email</i> check box is selected on the <i>Overview</i> tab, and the <i>Notify by Email</i> property for the Approval activity is set to true.</p>	

8.3 Log Activity

The Log activity is a system activity that writes messages to a log. To log information about the state of a workflow process, the Workflow System interacts with Novell® Audit.

NOTE: Novell Audit can be configured to send its information to Novell® Sentinel for additional logging and reporting features (see “[Logging to a Novell Audit or Sentinel server](#)” in the *Identity Manager 3.5 User Application: Administration Guide*).

During the course of its processing, a workflow can log information about various events that have occurred. Users can then use the Novell reporting tools to look at logged data.

Before you can use logging, you must enable logging in the user application. For more information see “[Enabling Audit Logging](#)” in the *Identity Manager 3.5 User Application: Administration Guide*.

NOTE: During the course of workflow execution, many system events are logged that are not controlled by the Log Activity. For example, the Workflow System writes a message to the log whenever a workflow is started or stopped, or when it is approved, denied, or refused. For a complete list of the system events logged during workflow execution, see “[Setting Up Logging](#)” in the *Identity Manager 3.5 User Application: Administration Guide*.

8.3.1 Properties

The Log activity has the following properties:

Table 8-6 Log Activity Properties

Property Name	Description
<i>Name</i>	Provides a name for the activity.
<i>Audit</i>	Specifies whether log messages should be sent. When this property is set to true, messages are sent to all log4j channels, including Novell Audit. When this property is set to false, no log messages are sent.
<i>Author</i>	Defines the author for the message. By default, the author is the initiator of the provisioning request.
<i>Message</i>	<p>Specifies an ECMA expression that defines text for the log message. Typically, this text indicates where this Log activity is being executed within the process and provides other information that makes the log easy to understand.</p> <p>For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 249. For descriptions of the system variables available in a workflow, see Section 4.3.3, “Understanding Workflow Data,” on page 86.</p>

8.3.2 Data Item Mapping

Not supported with this activity.

8.3.3 E-mail Notification

Not supported with this activity.

8.4 Branch Activity

In a workflow that supports parallel processing, the Branch activity allows multiple users to act on different areas of the work item in parallel. After the users have completed their work, the Merge activity synchronizes the incoming branches in the flow.

A workflow can have multiple Branch activities, but each Branch activity must have an associated Merge activity. All flow paths leading out of a Branch activity will execute.

The Branch activity does not support synchronization between the branches while they are executing. Each branch must not depend on data being updated in another branch. The data synchronization is enforced by the Merge activity. After the Merge activity completes, all of the data set in the branches is available.

8.4.1 Properties

The Branch activity has the following property:

Table 8-7 Branch Activity Properties

Property Name	Description
<i>Name</i>	Provides a name for the activity.

8.4.2 Data Item Mapping

Not supported with this activity.

8.4.3 E-mail Notification

Not supported with this activity.

8.5 Mapping Activity

The Mapping activity allows you to add or manipulate data in a workflow. It evaluates the source expression and saves the result in the target expression of the associated data items. You can use it as a way to combine data from parallel-processed approval forms after their data is moved to flowdata.

For example, in a parallel approval context you might need to collect data from more than one approval form that is dependent on each other or needs to be calculated with each other. To accomplish this, place a Mapping activity after a Merge activity and before any activities that consume the results (for example, Condition, Entity, Provisioning or another Approval activity).

You can also use the Mapping activity to isolate calls to external Java routines that might manipulate data and be resource intensive, thereby not slowing down user-based Approval activities in either their pre-activity or post-activity mapping phase.

8.5.1 Properties

The Mapping activity has the following property:

Table 8-8 Mapping Activity Properties

Property Name	Description
Name	Provides a name for the activity.

8.5.2 Data Item Mapping

To bind the data items associated with the Mapping activity, you define pre-activity and post-activity mappings. The pre-activity mappings initialize data in flowdata with constants, values retrieved from the flowdata object, system process variables, system activity variables, or data retrieved via expression calls to the directory abstraction layer. The post-activity mappings move data into the flowdata object.

Table 8-9 Mapping Activity Data Item Mappings

Setting	Description
Source Expression	Specifies a source expression. When you click a cell in the <i>Source Expression</i> column, the ECMA expression builder displays to help you define your expression. For example, <pre>function list() { s=new java.lang.String(); if (wi.XPath('count(flow-data/groups)') > 0) s="There was a group selected"; return s;}; list();</pre>
Target Expression	Specifies a target expression. When you click a cell in the <i>Target Expression</i> column, the ECMA expression builder displays to help you define your expression or you can click the Map All button. An example of a target expression is: <pre>flowdata.testexpression</pre>

8.5.3 E-mail Notification

Not supported with this activity

8.6 Merge Activity

In a workflow that supports parallel processing, the Merge activity synchronizes the incoming branches in the flow. The Merge activity is used in conjunction with the Branch activity, which allows two users to act on different areas of the work item in parallel. After the users have completed their work, the Merge activity synchronizes the incoming branches.

A workflow can have multiple Branch activities, but each Branch activity must have an associated Merge activity.

8.6.1 Properties

The Merge activity has the following property:

Table 8-10 *Merge Activity Properties*

Property Name	Description
<i>Name</i>	Provides a name for the activity.

8.6.2 Data item mapping

Not supported with this activity.

8.6.3 Email notification

Not supported with this activity.

8.7 Condition Activity

The Condition activity lets you add conditional logic to a workflow. This logic can be used to control what happens when the workflow executes. In the Condition activity, you define logic as an ECMA expression that evaluates to a boolean value.

Each Condition activity must have two outgoing flow paths, one that handles conditions that evaluate to true and another that handles conditions that evaluate to false. Optionally, a third flow path can be added to handle error conditions that occur if the ECMA expression evaluation fails.

8.7.1 Properties

The Condition activity has the following properties:

Table 8-11 *Condition Activity Properties*

Property Name	Description
<i>Name</i>	Provides a name for the activity.

Property Name	Description
<i>Condition Expression</i>	<p>Specifies an ECMA expression that returns true or false. The value returned determines which flow path is followed after the activity has finished executing.</p> <hr/> <p>TIP: If you need to test whether two objects are equal in a conditional expression, you should use the == operator, rather than the equals() method, unless you are certain that the objects being compared are Java objects of the same type. For instance, use this expression:</p> <pre>(approval_A.getAction() == "DENIED")</pre> <p>instead of this one:</p> <pre>(approval_A.getAction()).equals("DENIED")</pre> <hr/> <p>For details on building ECMA expressions, see Chapter 10, "Working with ECMA Expressions," on page 249. For descriptions of the system variables available in a workflow, see Section 4.3.3, "Understanding Workflow Data," on page 86.</p>

8.7.2 Data Item Mapping

Not supported with this activity.

8.7.3 Email Notification

Not supported with this activity.

8.8 Workflow Status

The Workflow Status activity lets you specify the approval status (approved or denied) for workflows that do not contain a provisioning activity (an Entitlement or Entity).

8.8.1 Properties

The Workflow Status activity has the following properties:

Table 8-12 *Workflow Status Activity Properties*

Property	Description
<i>Name</i>	Specifies the name of the activity.
<i>Workflow Status</i>	Specifies the approval status as an expression: either 'approved' or 'denied'.

8.8.2 Data Item Mapping

Not supported with this activity.

8.8.3 E-mail Notification

Not supported with this activity.

8.9 Finish Activity

The Finish activity marks the completion of a workflow. When the Finish activity executes, an e-mail message is sent to notify participants that the workflow has finished.

8.9.1 Properties

The Finish activity has the following properties:

Table 8-13 *Finish Activity Properties*

Property	Description
<i>Name</i>	Provides a name for the activity.
<i>Notify by Email</i>	<p>Provides a method of triggering an e-mail notification when the Finish activity is executed. When this property is set to true, an e-mail notification is sent. When this property is set to false, no e-mail notification is sent.</p> <p>See Section 8.9.3, “E-mail Notification,” on page 193 for information about setting up the e-mail notification.</p>

8.9.2 Data Item Mapping

Not supported with this activity.

8.9.3 E-mail Notification

To enable e-mail notification for the Finish activity, you need to specify the e-mail template to use, as well as source expressions for target tokens in the e-mail body.

Table 8-14 *Email Notification Settings for the Finish Activity*

Setting	Description
<i>Email Template</i>	<p>Specifies the name of the e-mail template to use. By default, the Finish activity uses the Provisioning Approval Completed Notification template.</p> <p>You can edit an e-mail template in Designer . See “Editing an e-mail template:” on page 164 for more information.</p>

Setting	Description
Source	Specifies the source expressions for target tokens in the e-mail body.
Target	<p>The list of target tokens is determined by the selected e-mail template. You cannot add new tokens, but you can assign values to the predefined tokens by building your own source expressions. At runtime, the source expressions are evaluated to determine the value of each token.</p> <p>The available target tokens for the Provisioning Approval Completed Notification e-mail template are listed below:</p> <ul style="list-style-type: none"> ♦ TO ♦ CC ♦ BCC ♦ requestStatus ♦ requestSubmissionTime ♦ requestID ♦ recipientFullName ♦ initiatorFullName ♦ requestTitle <p>If you use a provisioning request definition template to create your workflow, each token has a default source expression. The default expressions retrieve values from the workflow process (the process object) or from the data abstraction layer (IDVault object). You can modify these expressions to suit your application requirements.</p> <hr/> <p>NOTE: When you create a workflow for use with the Resource Request portlet (see “Resource Request Portlet” in the <i>Identity Manager 3.5 User Application: Administration Guide</i>) and you use <code>_default_</code> as the expression for the TO token as <code>_default_</code>, the addressee expression must be an IDVault expression.</p> <hr/> <p>For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 249.</p>
<p>NOTE: E-mail notification is only supported when the <i>Notify participants by email</i> check box is selected on the <i>Overview</i> tab.</p>	

8.10 Integration Activity

The Integration activity provides a way to use a Web service to process workflow data. For detailed information about using the Integration activity, see [Chapter 9, “Working with Integration Activities,”](#) on page 201.

8.10.1 Properties

The Integration activity has the following properties.

Table 8-15 *Integration Activity Properties*

Property Name	Description
<i>Name</i>	Provides a name for the activity.
<i>WSDL Resource</i>	<p>Specifies a WSDL file for the Web service to be used in the Integration activity. Once specified, the WSDL is incorporated into the provisioning request definition file.</p> <p>When you select a WSDL file, a dialog box is displayed that you use to select the Web service port type and operation that you want to use in the Integration activity.</p>
<i>Timeout</i>	<p>Specifies a dynamic expression that defines the period of time allotted for the Integration activity to complete. The timeout interval applies each time the activity is executed by the addressee.</p> <p>For details on building ECMA expressions, see Chapter 10, "Working with ECMA Expressions," on page 249. For descriptions of the system variables available in a workflow, see Section 4.3.3, "Understanding Workflow Data," on page 86.</p>
<i>Retry Count</i>	<p>Specifies the number of times to retry the activity in the event of a timeout.</p> <p>When an activity times out, the workflow process can try to complete the activity again, depending on the retry count specified for the activity. If the last retry times out, the activity can be marked as success, fault, error, or timed out, depending on the final timeout action specified for the activity.</p>
<i>Final Timeout Action</i>	<p>Determines the final state of the request in the event that the Integration activity times out. The choices are</p> <ul style="list-style-type: none">♦ success♦ fault♦ error♦ timedout

8.10.2 Data Item Mapping

To bind the data items associated with the Integration activity, you define pre-activity and post-activity mappings. The pre-activity mappings map values retrieved from the flowdata object to attributes in the Input message for the Web service that will be accessed by the Integration Activity. The post-activity mappings map the response from the Web service back into the flowdata object. For more information about data item mapping for Integration activities, see [Section 9.3, "Moving Data to and from the Integration Activity,"](#) on page 203.

Table 8-16 *Integration Activity Data Item Mappings*

Setting	Description
Pre-Activity	<p>Allows you to specify one or more pre-activity mappings. When this option is selected, you can double-click a cell in the <i>Source Expression</i> column to specify where the Integration activity gets data for a particular Web service input field.</p> <hr/> <p>NOTE: When the <i>Pre-Activity</i> option is selected, the cells in the <i>Web Service Input Field</i> column are not editable.</p> <hr/>
Post-Activity	<p>Allows you to specify one or more post-activity mappings. When this radio button is selected, you can double-click a cell in the Target Expression column to specify where data from a Web service output field should be copied after the form has been processed.</p> <hr/> <p>NOTE: When the <i>Post-Activity</i> option is selected, the cells in the <i>Web Service Output Field</i> column are not editable.</p> <hr/>
<i>Source Expression</i>	<p>Specifies a source expression. When you click a cell in the <i>Source Expression</i> column, the ECMA expression builder displays to help you define your expression. For example, <code>flowdata.get('Start/RequestRate/Country1')</code> for a Web service input, or <code>flowdata.Start/RequestRate/Country1</code> for a Web service output field.</p>
<i>Web Service Input Field</i>	<p>This column displays all of the input fields for the port type and operation specified when the WSDL file was selected. The fields in this column are automatically populated. If you want to remove an input field, click <i>Mapping</i>, expand the nodes of the sample document and de-select any input fields that you want to remove.</p>
<i>Web Service Output Field</i>	<p>This column displays all of the output fields for the port type and operation specified when the WSDL file was selected. The fields in this column are automatically populated. If you want to remove an output field, click <i>Mapping</i>, expand the nodes of the sample document and de-select any output fields that you want to remove.</p>
<i>Mapping</i>	<p>Displays a hierarchical view of the sample document for the inputs to or outputs from the Web service. You can use this feature to deselect input or output fields (by default, all Web service input and output fields are selected).</p>

8.10.3 E-Mail Notification

Not supported with this activity.

8.11 Entitlement Activity

The Entitlement activity grants or revokes an entitlement for a user or other entity type.

A workflow must have at least one Entitlement or Entity activity.

8.11.1 Properties

The Entitlement activity has the following properties:

Table 8-17 *Entitlement Activity Properties*

Property Name	Description
<i>Name</i>	Provides a name for the activity.
<i>Set Workflow Status</i>	Specifies the approval status of the provisioning request. Set to true for approved; otherwise, set to false. This method of setting workflow status overrides other methods (for example, the <i>Set Default Completion Status to Approved</i> parameter (see Table 5-1, “Overview Properties,” on page 101) or the <i>Approval Status</i> activity (see Section 8.8, “Workflow Status,” on page 192).

8.11.2 Data Item Mapping

To bind the data items associated with the Entitlement activity, you define mappings for several DirXML® attributes.

Table 8-18 *Entitlement Activity Data Item Mappings*

Setting	Description
Source Expression	<p>Specifies a source expression for a DirXML mapping. When you click a cell in the <i>Source Expression</i> column, the ECMA expression builder displays to help you define your expression.</p> <p>The DirXML mappings for the Entitlement are described below:</p> <ul style="list-style-type: none">• dn is the distinguished name for the recipient of the entitlement.• DirXML-Entitlement-DN is the distinguished name of the entitlement to execute. For example, the entitlement might be identified as follows: <code>'CN=Groups,CN=GroupEntitlementLoopback,CN=TestDrivers,O=novell'</code> You can use the ECMA expression builder's ECMAScript Variable panel to see a list of all the entitlements in the driver. To select an entitlement, double-click the full distinguished name of the entitlement.• DirXML-Entitlement-Action indicates whether the entitlement is granted or revoked. If the operation grants the entitlement, the value must be 1; if it revokes the entitlement, the value must be 0.• DirXML-Entitlement-Parameter specifies a parameter required by the entitlement driver. For example, if the entitlement operation grants access to the Sales group, the parameter might specify the group as follows: <code>'\\MYTREE\\novell\\idmsample-doc\\groups\\Sales'</code>• DirXML-Entitlement-MultiValueAllowed indicates whether the entitlement supports multiple values. If it supports multiple values, the value must be true; otherwise, it must be false.

For details on building ECMA expressions, see [Chapter 10, “Working with ECMA Expressions,” on page 249](#).

8.11.3 E-mail Notification

Not supported with this activity.

8.12 Entity Activity

The Entity activity updates an entity in the Identity Vault. You can use this activity to create, modify, or delete attributes on an entity. You can also use this activity to create or delete an entity (see [Section 7.7, “Working with Entity Activities,” on page 173](#)).

A workflow must have at least one Entitlement or Entity activity.

8.12.1 Properties

The Entity activity has the following properties:

Table 8-19 *Entity Activity Properties*

Property Name	Description
<i>Name</i>	Provides a name for the activity.
<i>Entity Type</i>	Specifies the target entity type: User or Group.
<i>Operation</i>	<p>Indicates what kind of operation will be performed on the target entity:</p> <ul style="list-style-type: none">♦ Create/Modify♦ Delete attributes/values♦ Delete entity <p>To create or modify attributes of an entity or to create a new entity, select <i>create/modify</i>. To delete attributes of an entity, select <i>delete</i>.</p> <p>To delete an entity, select <i>delete object</i>.</p>
<i>Set Workflow Status</i>	Specifies the approval status of the provisioning request. Set to true for approved; otherwise, set to false. This method of setting workflow status overrides other methods. For example, the <i>Set Default Completion Status to Approved</i> parameter (see Table 5-1, “Overview Properties,” on page 101) or the <i>Approval Status</i> activity (see Section 8.8, “Workflow Status,” on page 192).

8.12.2 Data Item Mapping

To bind the data items associated with the Entity activity, you define mappings for the attributes associated with the target entity type.

Table 8-20 *Entity Activity Data Item Mappings*

Setting	Description
Entity dn	<p>Identifies the entity that is the target of the operation. The default value is recipient.</p> <p>To create a new object, specify a distinguished name that does not yet exist.</p> <hr/> <p>TIP: The output of the DNMaker control can be used as input for the Entity dn value. The DNMaker control constructs the DN by allowing the user to enter the naming attribute in a text field and presenting an interface for picking a container. Once this data has been captured in a request form, the output can be mapped to a variable in the flowdata object. In the definition for the Entity activity, this flowdata variable can be accessed in the Entity dn setting with an expression such as <code>flowdata.get('groupdn');</code></p> <p>For details on using the DNMaker control, see Section 6.5.9, “DNMaker,” on page 133.</p> <hr/>
Modify Type	<p>Indicates how the mapping should be performed for an attribute. The choices are</p> <ul style="list-style-type: none">◆ Append Value◆ Replace Value◆ Replace All Values <p>For many attributes, <i>Replace Value</i> is the only option that makes sense; therefore, this option is selected automatically and cannot be changed.</p> <p>You must specify the <i>Modify Type</i> setting before specifying the <i>Modify Value Expression</i> setting.</p> <hr/>

Setting	Description
Modify Value Expression	<p>Specifies a source expression for an attribute. When you click a cell in the <i>Modify Value Expression</i> column, the ECMA expression builder displays to help you define your expression. The list of attributes available varies depending on which entity type was selected on the Properties tab.</p> <p>Designer automatically inserts a sample ECMAScript expression into this field. The code provided varies depending on the <i>Operation</i> property specified in Properties and the <i>Modify Type</i> selected in Data Item Mapping. For example, if you have specified Create/Modify for <i>Operation</i>, and Replace All Values for <i>Modify Type</i>, Designer inserts an expression that helps you to create a vector:</p> <pre>function list() { v=new java.util.Vector(); v.add('{Enter Item 1}'); v.add('{Enter Item 2}'); return v; }; list();</pre> <p>In some cases you might be able to create expressions that work as well or better than the sample expression. For example, instead of creating a vector for multiple attribute values, you can create a flowdata variable (see Section 4.3.3, “Understanding Workflow Data,” on page 86) to store multiple attribute values, and use the getObject function to retrieve the values of the flowdata variable (see “ECMAScript Variables” on page 252).</p> <p>NOTE: The cells in the <i>Target Attribute</i> column are not editable.</p>

8.12.3 Email notification

Not supported with this activity.

This section provides details about working with Integration activities. Topics include:

- ♦ [Section 9.1, “About the Integration Activity,” on page 201.](#)
- ♦ [Section 9.2, “Adding an Integration Activity,” on page 201](#)
- ♦ [Section 9.3, “Moving Data to and from the Integration Activity,” on page 203](#)
- ♦ [Section 9.4, “Using the Integration Activity Editor Interface,” on page 206](#)
- ♦ [Section 9.5, “Actions,” on page 218](#)

9.1 About the Integration Activity

The Integration activity is an activity that allows workflows to exchange data with arbitrary Web services. Data sent to a Web service can integrate an individual workflow with other systems, inside and outside the organization. Data received from a Web service can provide decision support information on approval forms.

You create flowdata variables to move data from the workflow to the Web service for processing. The Integration activity automatically creates an action model for working with a Web service based on a WSDL document that you specify.

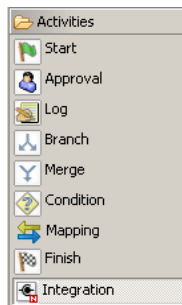
NOTE: The action model is a subset of the features available in the Novell Integration Manager product (formerly known as Novell exteNd Composer).

An action model is a visual representation of a set of instructions for processing XML documents and communicating with XML data sources. An action model performs all data mapping, data transformation, and data transfer within an Integration activity. You can edit the action model to manipulate data before and after the data is submitted to the Web service. You then map the data from the Integration activity back to flowdata variables for use in the workflow.

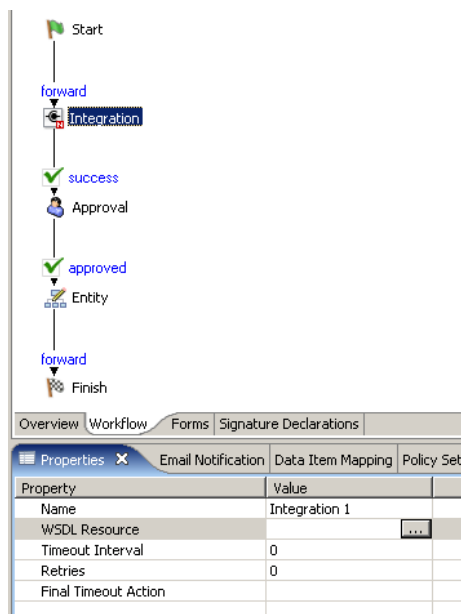
9.2 Adding an Integration Activity

- 1 Create a provisioning request definition (see [Chapter 5, “Creating a Provisioning Request Definition,” on page 95](#)).
- 2 Create a workflow for the provisioning request definition (see [Chapter 7, “Creating the Workflow for a Provisioning Request Definition,” on page 157](#)).
- 3 Click the *Workflow* tab.

- 4 Drag an Integration activity from the palette and place it in the desired location in the workflow.

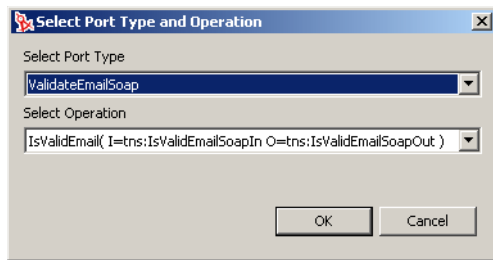


- 5 Click the *Properties* tab.



- 6 Type a name for the activity in the *Name* field.
- 7 Click the *Value* field for the *WSDL Resource* property, then click the browse button to display a dialog box that you use to locate the WSDL file for the Web service that you want to access with the Integration activity.
- 8 Use the dialog box to browse your file system to locate the WSDL file for the Web service that you want to use. Click the name of the WSDL file, then click *Open* to return to the *Properties* tab.

A dialog box that you use to select a port type and operation for the Web service is displayed.



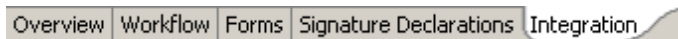
The *Select Port Type* list includes a set of port types supported by the Web service. Each port type supports operations that include the input and output messages of the operation.

- 9 Select a port type from the list.
- 10 Select an operation from the *Select Operation* list.
- 11 Click *OK*.

The Integration activity creates an action model based on the WSDL document. You use the action model at design time to test the input to the Web service, test the response from the Web service, and map and transform data, if necessary, before returning the data to the workflow.

For many Web services, you won't need to concern yourself with the action model. You will simply create data item mappings for the Integration activity. After the action model is created, a new tab, *Integration*, is added to the provisioning request definition editor. You use this tab to access the action model.

- 12 Specify the *Timeout Interval*, *Retries*, and *Final Timeout Action* properties (see [Section 8.10, "Integration Activity,"](#) on page 194).
- 13 If you want to view or edit the action model, click the *Integration* tab.



9.3 Moving Data to and from the Integration Activity

- 1 Create form fields to allow users to provide input to the Web service accessed by the Integration activity (see [Chapter 6, "Creating Forms for a Provisioning Request Definition,"](#) on page 103). For example, if you are working with a Web service that provides stock quotes, you will need a field for the user to specify a stock symbol.
- 2 To move user input from the form to the workflow, create a flowdata variable in an activity that precedes the Integration activity in the workflow.

See [Section 4.3.3, "Understanding Workflow Data,"](#) on page 86 for information about creating flowdata variables.

For example, if you have created a form field called "symbol" to accept a stock symbol for input to the Web service, you would go to the post-activity data item mapping for the activity associated with the form that contains the *symbol* field, then you would map the *symbol* field to a flowdata variable (for example, flowdata.symbol).

- 3 In the *Workflow* tab, right-click the icon for the Integration activity, then choose *Show Data Item Mapping*.

The *Data Item Mapping* tab is displayed.

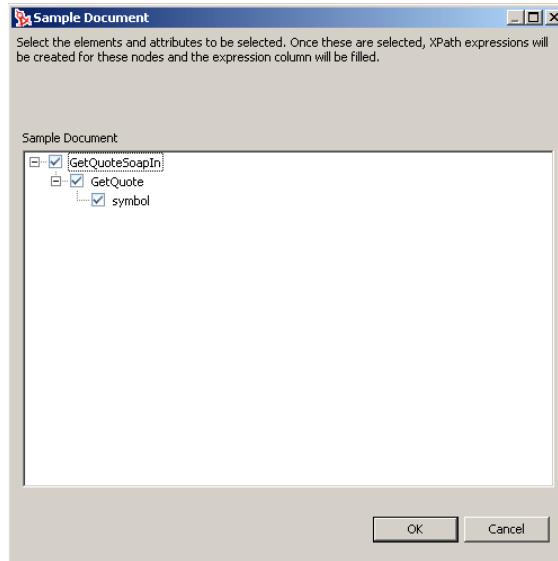
- 4 In the *Data Item Mapping* view, click *Pre-Activity*.

In the *Web Service Input Field* grid, you should see fields that match all of the input fields associated with the port type and operation specified in [Step 9](#) and [Step 10 on page 203](#).

- 5 The Integration activity automatically selects all of the input fields associated with the port type and operation. If you would like to remove some of the input fields, follow these steps:

5a Click *Mapping*.

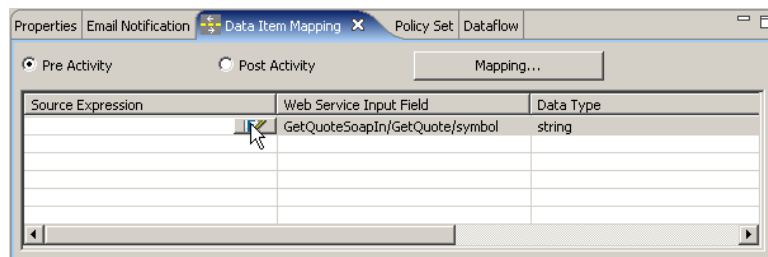
The *Sample Document* dialog box is displayed.



- 5b Expand the nodes of the sample document and de-select any input fields that you want to remove.

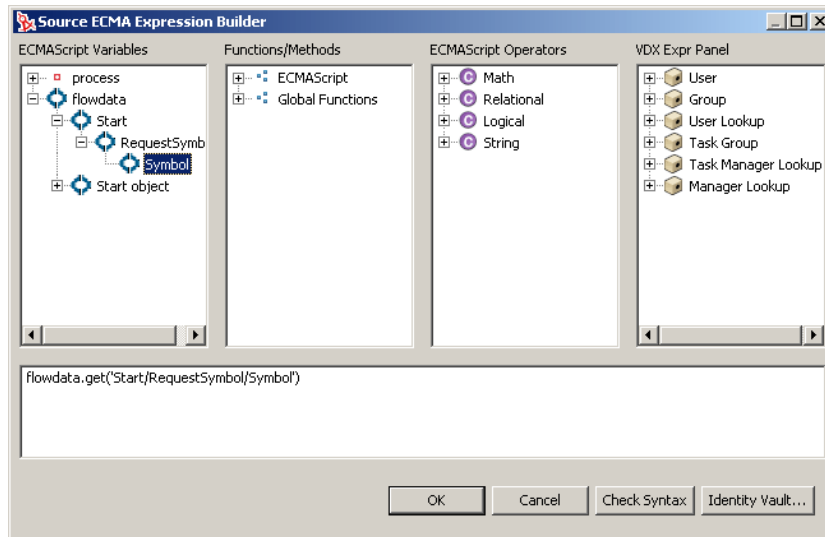
5c Click *OK* to return to the *Data Item Mapping* view.

- 6 For each *Web Service Input Field*, click in the *Source Expression* field, then click the ECMA expression builder button.



The ECMA expression builder is displayed.

- 7 Expand the *flowdata* node in the *Ecmascript Variables* pane of the ECMA expression builder, then double-click the flowdata variable for the user input to the Web service.



- 8 Click *OK* to return to the *Data Item Mapping* view.

- 9 Click *Post Activity*.

In the *Web Service Output Field* grid, you should see fields that match all of the output fields associated with the port type and operation specified in [Step 9](#) and [Step 10 on page 203](#).

- 10 The Integration activity automatically selects all of the output fields associated with the port type and operation. If you would like to remove some of the output fields, follow these steps:

- 10a Click *Mapping*.

The *Sample Document* dialog box is displayed.

- 10b Expand the nodes of the sample document and de-select any attributes that you want to remove.

- 10c Click *OK* to return to the *Data Item Mapping* view.

- 11 Click *Map All* to automatically create flowdata variables for each *Web Service Output Field*.

Alternatively, for each *Web Service Output Field*, click in the *Source Expression* field, then click the ECMA expression builder button.

- 12 Expand the *flowdata* node in the *Ecmascript Variables* pane of the ECMA expression builder, then double-click the flowdata variable that will receive data from the Web service.

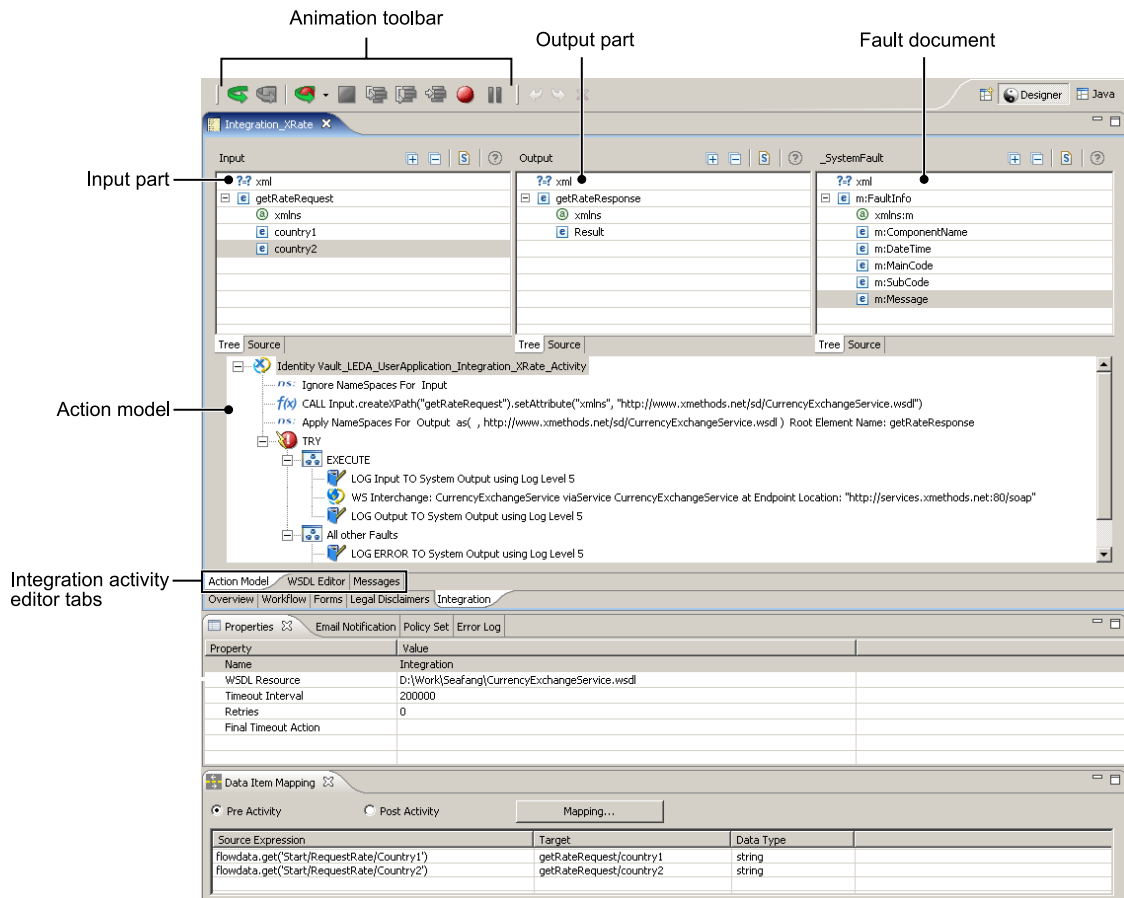
- 13 Click *OK* to close the ECMA expression builder.

The next step is to work in the Integration view to test and refine the interaction with the Web service.

9.4 Using the Integration Activity Editor Interface

The Integration activity editor provides a working environment for the input, output, and actions of the Integration activity. The Integration activity editor is composed of three views: Action Model, WSDL Editor, and Messages.

Figure 9-1 *Integration Activity Interface*



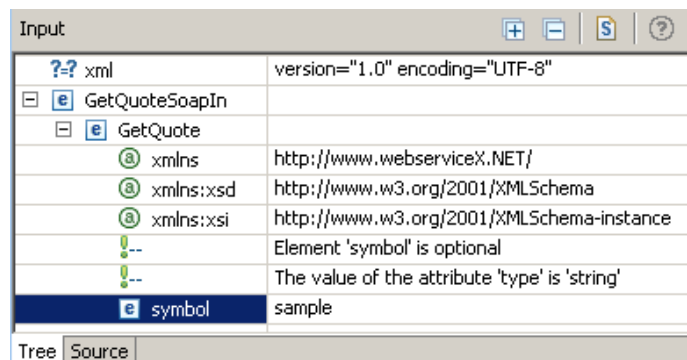
9.4.1 XML Views

The Integration activity provides a number of XML views (for example, Input and Output messages, WSDL Editor, Messages) derived from the WSDL document. These views use a common interface.

Tree View

You use the Tree view to work with a hierarchical view of an XML document. You display the Tree view by clicking the *Tree* tab.

Figure 9-2 *Tree View*



Tree View Editing Features

The Tree view provides the following editing features:

- ◆ You can edit attribute values, attribute names and values, namespace names and values, text, and comments.
- ◆ You can insert new nodes using the menu that is displayed when you right-click within the Tree view. The menu allows you to insert nodes as children before or after the selected node. If the node is an element, you can insert attributes. The submenus for Add Child, Add After, and Add Before contain the node that can be legally added. If no schema or DTD is associated with the document, the submenus contain New Attribute or New Element.
- ◆ You can delete a node by right-clicking a node and selecting *Remove*.
- ◆ You can drag and drop items between Tree views (for example, between views of the Input and Output messages) to create Map actions (see [Section 9.5.8, “Map,” on page 240](#) for information about Map actions).
- ◆ You can undo, redo, cut, copy, and paste.

Tree View Menu

When you right-click an item in the Tree view, a menu is displayed that you use to perform operations on the XML document. The menu is context-sensitive and only displays the commands that are appropriate for the item on which you clicked.

Table 9-1 *Tree View Menu*





Item	Description
Remove	Removes the selected item.
Add DTD Information	Displays a dialog box that you use to add DTD information. You can edit the <i>Root Element Name</i> , <i>Public ID</i> , and <i>System ID</i> .
Edit Namespaces	Displays a dialog box that you use to add namespace declarations.

Item	Description
Add Attribute	Displays a dialog box that you use to define a new attribute.
Add Child	Displays a submenu with the following options:
Add Before	Comment
Add After	Add Processing Instruction #PCDATA CDATA Section New Element
Replace with	Replaces the selected item with an item selected from the menu.

Tree View Toolbar

Tree view toolbars provide the following features:

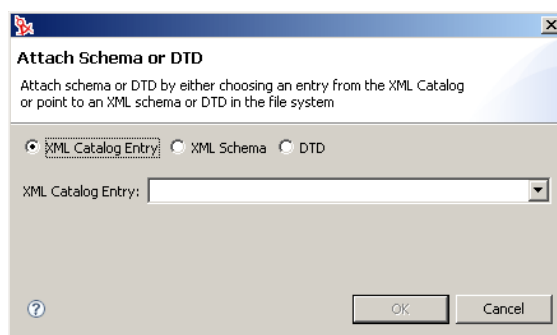
Table 9-2 Enter Table Title Here

Button	Description
	Expands all nodes in the document.
	Collapses all nodes in the document.
	Attaches a schema or DTD (see “Attaching a Schema or DTD” on page 208).
	Displays online help.

Attaching a Schema or DTD

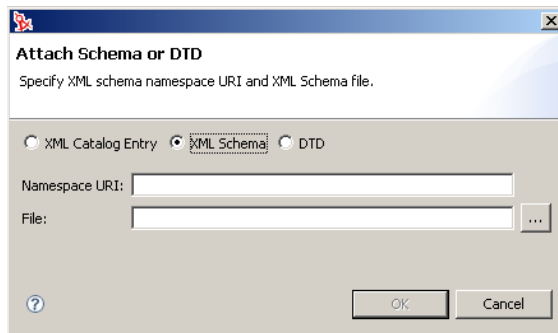
You can attach a schema or DTD to the current XML document when you are using the Tree view.

- 1 Click  in the Tree view toolbar. The *Attach Schemas or DTD* dialog box is displayed.

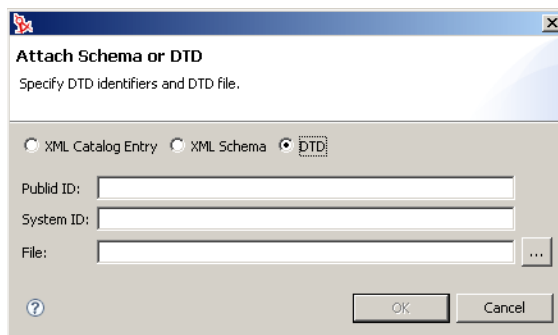


- 2 To choose from a list of entries in the XML catalog, choose an entry from *XML Catalog Entry* list.

- 3 To specify an XML schema on disk, click *XML Schema*.



- 4 Type a *Namespace URI*, then use the browse button in the *File* field to select an XML schema on disk.
- 5 To specify a DTD on disk, click *DTD*.

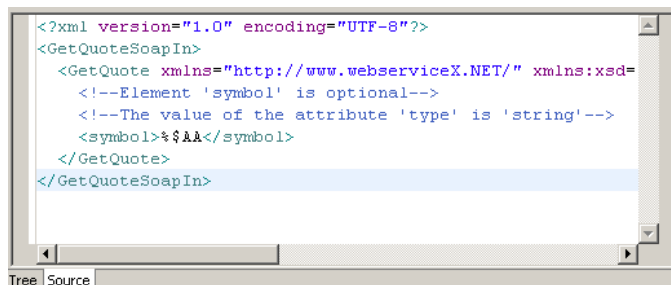


- 6 Type a *Public ID* and *System ID*, then use the browse button in the *File* field to select the DTD file on disk.

Source View

You use the Source view to view the XML source of the document. You display the Source view by clicking the *Source* tab.

Figure 9-3 Source View



Source View Features

The source view supports the following features:

- ◆ Syntax highlighting.

- ♦ Context-sensitive code-completion based on DTD or XML Schema.
- ♦ Validation as you type. If the XML is invalid (for example, the closing bracket is omitted from a tag), the editor indicates the error.
- ♦ General text editing operations such as undo, redo, cut, copy, paste, and select all.

Source View Menu

When you right-click an item in the Source view, a menu is displayed that you use to perform operations on the XML document.

Table 9-3 *Source View Menu*

Item	Description
Undo	Reverse the last action.
Redo	Reverse an undo operation.
Cut	Cut the selected text to the clipboard.
Copy	Copy the selected text to the clipboard.
Paste	Paste the clipboard contents at the insertion point.
Delete	Deletes the selected text.
Select All	Selects all of the text in the document.
Find	Displays a dialog box that you use to find and replace text within the document.

9.4.2 Action Model

The action model includes the Action Model view and views for displaying message parts. The Action Model view displays actions that operate on the contents of the message parts. The message parts display the XML for the Web service Input and Output messages.

About the Action Model Views

The action model views are used at design time to test the interaction with the Web service. You edit actions in the Action Model view. You can enter test data to be input to the Web service in the Input view, examine the response from the Web service in the Output view, and see any error messages returned from the Web service in the _SystemFault view. The Integration activity has the following message panes:

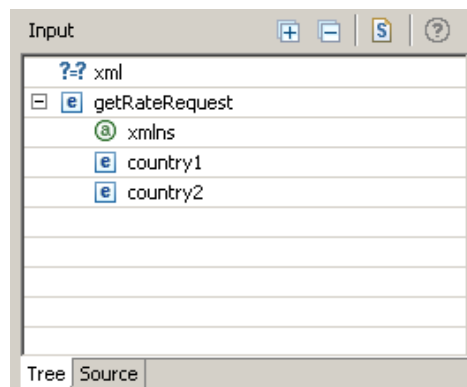
- ♦ Input view
- ♦ Output view
- ♦ _SystemFault view
- ♦ Action Model view

About the Input View

The Input view displays the input message derived from the WSDL document for the Web service. You can resize the view by dragging the right-hand border. You can resize columns within the view

by dragging the column border. You can specify a value to use in testing the action model directly in the Input part, in which case the value is discarded after executing the action model. You can also specify a value using the *Messages* tab (see [Section 9.4.4, “Messages,” on page 217](#)), in which case the value persists until you delete the value or you regenerate the action model (see [Section 9.4.5, “Regenerating Code for the Action Model,” on page 217](#)).

Figure 9-4 *Input View*

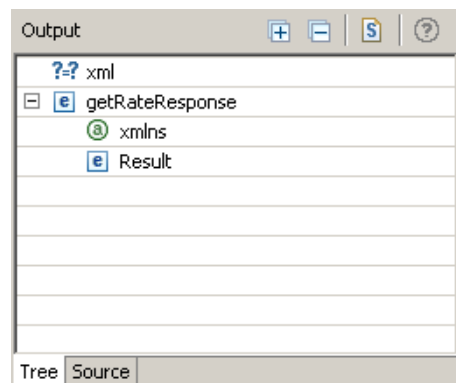


About the Output View

The Output view displays the output message derived from the WSDL document for the Web service. When you execute the action model, you use the Output view to view the values returned from the Web service.

You can resize the view by dragging the left-hand border. You can resize columns within the view by dragging the column border. You can specify a value directly in the Output part for modeling purposes, in which case the value is discarded after executing the action model. You can also specify a value using the *Messages* tab (see [Section 9.4.4, “Messages,” on page 217](#)), in which case the value persists until you delete the value.

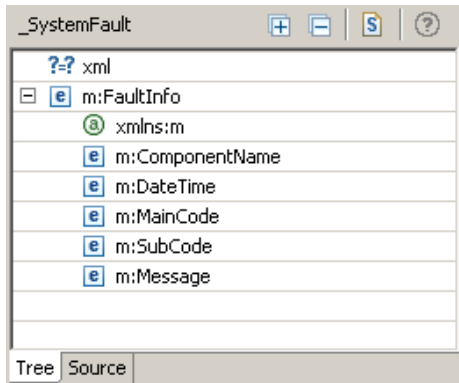
Figure 9-5 *Output View*



About the _SystemFault View

The _SystemFault view displays any error messages produced when you execute the action model. The XML information contained in _SystemFault also gets written to a global object called ERROR.

Figure 9-6 _SystemFault View



Beneath the FaultInfo root are the following elements:

- ♦ *DateTime* contains the Date and Time at which the fault occurred.
- ♦ *ComponentName* contains the name of the component that threw the fault.
- ♦ *MainCode* contains the main code number for the error.
- ♦ *SubCode* contains a sub-code number for the error.
- ♦ Message contains the error message defined when you set up a Throw Fault action (see **“Throw Fault” on page 220**). If you do not specify an error message in your Throw Fault action, the following message is displayed, “A user-defined Fault occurred!”. If the error occurred within a Try/On Fault action, and you did not specify a Fault, this element is populated with an Exception message.

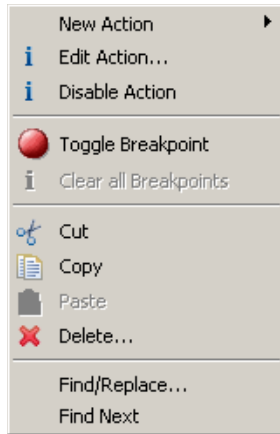
About the Action Model Pane

The Integration activity has a single action model. The action model represents the mappings, transformations, and other actions that will be performed on the Web service input and output messages. The Action Model view is resizable. Most of your activity that takes place in the Action Model view involves adding and editing actions.

Action Model Context Menu

If you right-click in the action model, a menu is displayed.

Figure 9-7 Action Model Menu

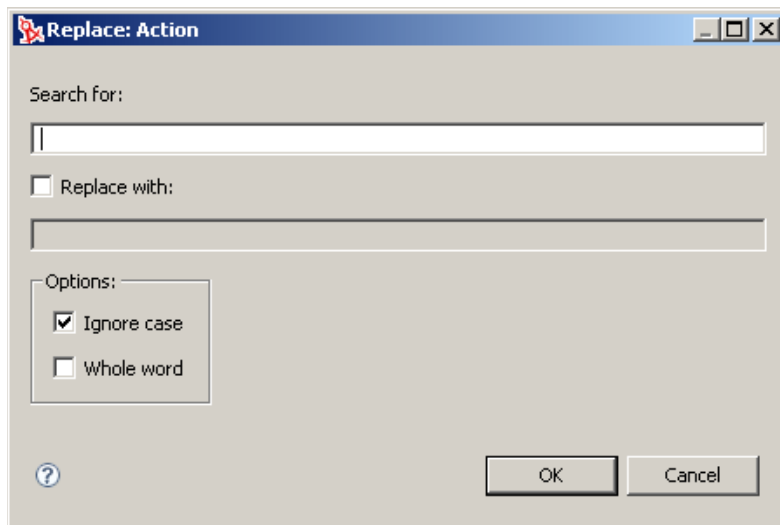


From this menu, you can add or edit actions (see [Section 9.5, “Actions,” on page 218](#)), toggle breakpoints in the action model (see [“Animation” on page 214](#)) and perform other tasks.

Finding and Replacing Text in the Action Model

You can replace a word or string using the Replace command on the action model menu.

- 1 Right-click in the action model, then select Replace.



- 2 Enter the search text.
- 3 If you want to replace the search text, click *Replace with*, then type a string to replace the search string.
- 4 If you want to find the search text regardless of the capitalization of the text, click *Ignore case*.
- 5 If you want to find the search text in whole words only, click *Whole word*.
- 6 Click *OK*

The Integration activity finds the first occurrence of the search text. If the operation is a find and replace operation, the Integration activity asks you to confirm the replacement. You can then replace the next or all occurrences of the search text.

Animation

The action model provides animation tools that you can use to test and troubleshoot actions interactively within the Integration activity. You can execute the action model step by step and watch the result of each action. Not only will you see any errors as they happen, but you can verify, in real time, that connections and data behaved as you planned.

The animation tools allow you to toggle one or more breakpoints. You can use this feature to concentrate on a particular section of an action model. When used in conjunction with the run-to-breakpoint tool, breakpoints allow you to quickly run through action model sections that work properly, coming to a stop at a particular action. From there, you can step through each action in sequence. You can also step over loops and other code blocks that would otherwise be tedious to execute step-by-step.






The Basic Animation Tools





The animation tools are available on the Designer toolbar.

Figure 9-8 Animation Toolbar



Table 9-4 Animation Tools

Animation Toolbar Button	Name	Description
	Execute	Executes the action model.
	Execute Current Action	Executes the currently selected action.
	Start Animation	Starts the animation process. Enables <i>Step Into</i> , <i>Step Over</i> , and <i>Run to Breakpoint/End</i> .
	End Animation	Stops the animation process.
	Step Into	<p>Executes the currently selected action and highlights the next sequential action.</p> <p>For a Repeat Loop action, pressing Step Into executes each action in the loop and iterates through each loop.</p> <p>For a Decision Action, Step Into processes the next action in the True or False branch.</p> <p>For the Try/On Fault action, Step Into processes the next action inside the execute branch, and possibly the On Fault branch.</p>

Animation Toolbar Button	Name	Description
	Step Over	Executes the currently selected action and highlights the next sequential action. Unlike the Step Into button, clicking this button does not highlight and execute the details of Repeat, Decision, or Try/On Fault actions.
	Run To Breakpoint/End	Runs the animation to the next breakpoint or to the end of the action model if there are no breakpoints.
	Toggle Breakpoint	Sets the highlighted action in the action model as a breakpoint. You may set more than one breakpoint. Another way to toggle a breakpoint is to right-click the desired action and select <i>Toggle Breakpoint</i> from the menu.
	Pause Animation	Pauses the animation.

Starting Animation

When you first open an Integration activity, *Start Animation* and *Toggle Breakpoint* are the only enabled buttons. When you click *Start Animation*, the rest of the animation buttons are enabled. If you want to halt the animation temporarily, you can use the *Pause Animation* button. If you want to abort the animation, you can do so at any time by clicking *End Animation*.

Although Copy, Paste, and action editing operations (including adding new actions) are all available at animation time, we recommend that you do not edit the action model during animation. If you do, exceptions or unpredictable behavior may occur. If you need to edit the action model, use *End Animation* to stop the animation first. Then apply your edits and begin the animation again.

- 1 Open an Integration activity.
- 2 Click *Start Animation* button in the Designer toolbar. All of the animation buttons become active except the Start Animation button, which is now dimmed.
- 3 Follow the instructions in the following sections to perform the desired Animation activity.

Toggling a Breakpoint

You use the Toggle Breakpoint tool to set a breakpoint in the action model where you want the animation process to stop. This is helpful if you have a lengthy action model with long sections that work properly. You can set the breakpoints for each action that is causing a problem and then step through the action to troubleshoot it.

- 1 In the Action pane, select the action for which you want to set a breakpoint. This is where the animation will stop.
- 2 Click *Toggle Breakpoint* on the Designer tool bar, or right-click the action and select *Toggle Breakpoint*. A dot appears in the left-hand border of the action model to indicate the breakpoint.
- 3 If desired, repeat the previous steps to select additional Breakpoints.

Stepping Into an Action

Step Into runs the highlighted action in the action model and then moves to the next action in the sequence. You can use the Step Into tool to step through each action in the entire action model, or

you can use it in conjunction with the Run to Breakpoint tool. Execution stops at the next breakpoint or when the action model ends, whichever comes first.

A possible scenario for using a breakpoint would be if you have ten actions that you know work properly but have doubts about the eleventh. You could set the eleventh action as a breakpoint, execute the Run to Breakpoint tool, and then step through the eleventh (and subsequent) actions by executing the Step Into tool.

- 1 Start the animation (see “Starting Animation” on page 215).
- 2 Click *Step Into*. The first action in the action model is highlighted.
- 3 Click *Step Into* again. The highlighted action executes and the next action becomes highlighted.
- 4 Continue to work through the action model by clicking *Step Into* after each action executes and the subsequent action becomes highlighted.

Stepping Over an Action

You use *Step Over* when you don’t want to step into the details of the Repeat, Decision, or Try/On Fault actions. You can execute an entire block of code without stepping individually through each action.

You can use Step Over in conjunction with Run to Breakpoint. For example, you can toggle a Breakpoint, execute Run to Breakpoint, and then use Step Over to execute the action designated as the breakpoint. Step Over can save a great deal of time when testing lengthy action models, since you can avoid tediously stepping through individual actions.

- 1 Start the animation (see “Starting Animation” on page 215).
- 2 Step through the action model with *Step Into* until you reach a loop or other line of code that precedes an indented code block.
- 3 Click *Step Over*. The first action after the block of indented code becomes highlighted. All of the indented code executes normally and you are taken straight to the next block of actions, without stepping through the indented actions.
- 4 Continue to work through the action model by clicking *Step Over* as needed.
- 5 Continue to click *Step Into* and *StepOver* to execute all of the actions in the action model.

Pausing Animation

You use *Pause Animation* to pause the execution of an action in the action model. This is especially helpful in cases in which the action model contains lengthy loops.

- 1 During the execution of an action, click *Pause Animation*.
- 2 To resume the animation process, click *Step Into*, *Step Over*, or *Run to Breakpoint* (if a breakpoint has been set).

Aborting Animation

You use *Stop Animation* to stop the animation process. Once you stop the animation, you cannot restart from the place where you left off. You must restart from the beginning of the action model.

9.4.3 WSDL Editor

The *WSDL Editor* displays the WSDL document for the Web service. You can edit the WSDL using the *Tree* view and *Source* view editing features (see “[Tree View](#)” on page 207 and “[Source View](#)” on page 209).

9.4.4 Messages

The Messages view displays the messages derived from the WSDL document for the Web service. You can edit the messages using the Tree view and Source view editing features (see “[Tree View](#)” on page 207 and “[Source View](#)” on page 209). You can use this feature for entering test data that will be used when you execute the action model at design time. Data that you enter in the *Messages* view persists across executions of the action model.

9.4.5 Regenerating Code for the Action Model

When working in the *WSDL Editor* view, you can regenerate all code for the action model and regenerate messages by clicking the regenerate button. When working in the *Messages* view, you can regenerate all actions in the action model. The regenerate button is located in the Designer toolbar:

Figure 9-9 Regenerate Button



9.4.6 Adding Actions to the Action Model

Actions are the processing steps that take place within the Integration activity. A collection of actions is referred to as an action model. An action in the action model is displayed as a line and contains an icon for the action type along with an abbreviated definition of the action. Some actions are subordinate to other actions. For example, you can create a Repeat action that controls loop processing, then add actions inside the loop. The actions inside the loop are subordinate to the Repeat action and appear indented beneath the Repeat action. Subordinate actions process as long as the Repeat action is true.

To add an action to the action model, click the line in the action model that is one line above the position in which you want to insert an action. Add an action using any of following methods. The new action is inserted below the line you highlighted.

- ♦ Drag and drop. You can add Map actions by dragging and dropping elements from the *Input* view to the *Output* view.
- ♦ Copy and Paste. You can copy an action in the *Action Model* view and paste it somewhere else in the view.
- ♦ Right-click the line in the action model that is one line above the position in which you want to add the action, then select the desired action from the menu.

NOTE: You can reorder actions in the action model by dragging actions to a new position within the action model.

After you have created the action model, and before you deploy, you should test the action model. Perform testing by using the Animation tools. With the Animation tools, you can set breakpoints, start an animation, step into and over actions, and pause the animation. See [“Animation” on page 214](#).

9.5 Actions

This section describes the actions that are available for use within an action model. An action is similar to a programming statement in that it takes input in the form of parameters and performs specific tasks. An action model is a set of instructions for processing XML documents and communicating with XML data sources. An action model performs all data mapping, data transformation, and data transfer within an Integration activity. All actions within an action model work together.

At runtime, every action is converted to an executable form of ECMAScript and processed. At design time, many actions accept ECMAScript expressions as parameters, adding great flexibility and control to the action model. The Function action provides you with the most flexibility and control by giving you access to the full functionality of the ECMAScript language.

This section contains the following topics:

- ◆ [Section 9.5.1, “Advanced,” on page 218](#)
- ◆ [Section 9.5.2, “Data Exchange,” on page 224](#)
- ◆ [Section 9.5.3, “Repeat,” on page 229](#)
- ◆ [Section 9.5.4, “Comment,” on page 236](#)
- ◆ [Section 9.5.5, “Decision,” on page 237](#)
- ◆ [Section 9.5.6, “Function,” on page 238](#)
- ◆ [Section 9.5.7, “Log,” on page 239](#)
- ◆ [Section 9.5.8, “Map,” on page 240](#)

9.5.1 Advanced

This section includes descriptions of the following actions:

- ◆ [“Apply Namespaces” on page 218](#)
- ◆ [“Throw Fault” on page 220](#)
- ◆ [“Try/On Fault” on page 222](#)

Apply Namespaces

This section includes the following topics:

- ◆ [“About the Apply Namespaces Action” on page 218](#)
- ◆ [“Creating an Apply Namespaces Action” on page 219](#)

About the Apply Namespaces Action

Ideally, an Integration activity will always receive valid XML documents (that is, the documents validate against their schema, map and transform data appropriately, and send valid XML

documents). However, this might not always be the case. In other cases you might want to ignore namespaces altogether. It is important to have some means of validating XML documents. These and many other XML processing cases require a method of modifying or overriding the prefix and namespace handling provided by the Integration activity.

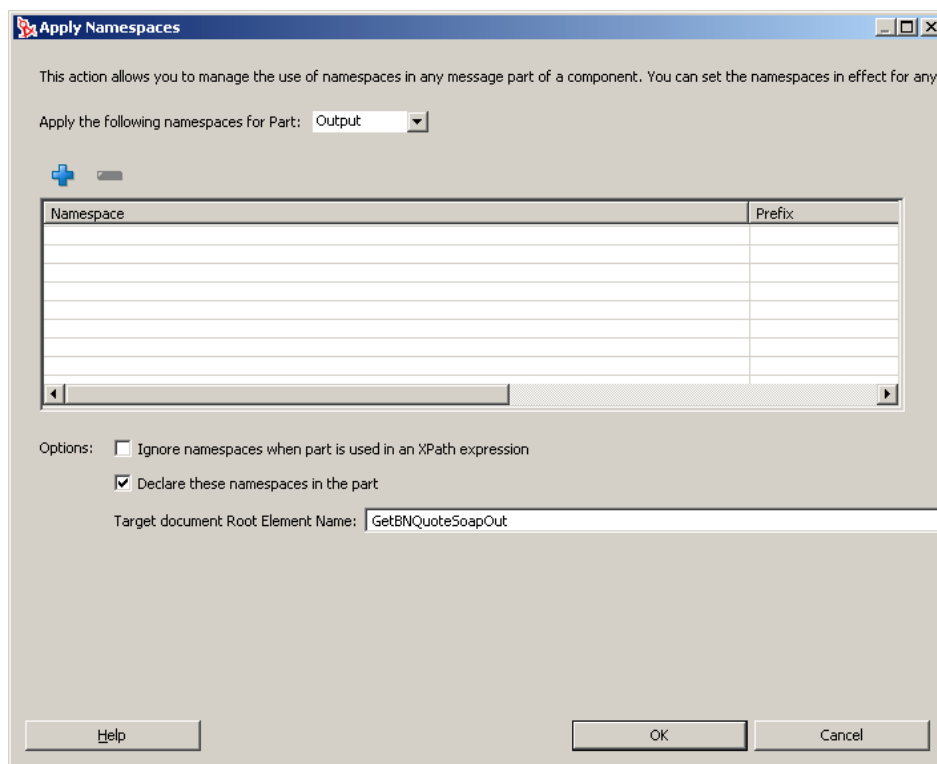
The Apply Namespaces action provides a mechanism for managing namespaces and namespace prefixes in effect for input and output messages within an action model. The action allows you to consolidate namespace and prefix declarations for a Web service in one place, to override those declared in the input and output messages, or to ignore namespaces altogether.

The Apply Namespaces action can be applied to input and output messages. You can also have multiple Apply Namespaces actions for an individual message part, effectively changing namespaces based on conditions specified in the action model. The namespaces declared are in effect until the end of the action model is reached or another Apply Namespaces action is executed. In other words, only the most recent Apply Namespaces action is in effect.

When creating a new Integration activity, an Apply Namespaces action is created automatically for the Output message if the WSDL declares any namespaces. After component creation, you can manually create additional Apply Namespaces actions.

Creating an Apply Namespaces Action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the Apply Namespaces action (the new action will be inserted below the line that you selected).
- 2 Select *New Action > Advanced > Apply Namespaces*.



- 3 Select the message (Input, Output, _SystemFault, or Project) to which you want to apply the namespace from the *Apply the following namespaces for Part* list.

- 4 Click the plus (+) icon to add a new row, then click the *Namespace* column and type a namespace URI.

The table displays all the Namespace declarations for the selected message part. After creating a new Apply Namespaces action, the table might or might not contain a list of declarations for a selected part. The list of declarations is initially constructed from the declarations defined in the WSDL.

Within the declaration list for a message part, prefixes must be unique. However, you can have duplicate namespace URIs provided that the URIs have unique prefixes. This allows you to declare multiple prefixes that are associated with the same namespace URI.

- 5 If desired, click *Ignore Namespaces when document is used in an XPath expression*.

Use this option when you want Map action source XPath expressions to find elements by their XML local name only.

This provides for a less restrictive method of specifying Map actions (see [Section 9.5.8, “Map,” on page 240](#)) and is useful when Map actions contain the wrong prefix or no prefix in their Source specifications. This allows you to place the Apply Namespaces action inside a Decision action (see [Section 9.5.5, “Decision,” on page 237](#)) that tests whether the Input message contains prefixes or not, yet still have one set of Map actions to map the input to another part. In other words, the Integration activity normally expects the input to contain prefixes, so you design all your Map actions with prefix names. For the occasional input that has no prefixes, the Decision action activates the Apply Namespaces action defined to ignore namespaces for input, allowing the Map actions to work in either case.

- 6 When you want to declare a set of namespaces in the root element of an output message built by your action model, click *Declare these namespaces in the part*.

This option is almost always checked for output to ensure that prefixed elements created in the output, as a result of Map actions, will resolve to the proper namespaces.

This allows a recipient of the output to validate the document properly. The Apply Namespaces action with this option checked could also be used to add new declarations to an existing document that already contains declarations.

The *Target document Root Element Name* is used to specify the name of the root element to contain the Namespace declaration attributes. The Integration Activity automatically fills in this value based on the information in the WSDL document and the message part specified in the Apply the following namespaces for Part list.

- 7 Click *OK*. The new action is added to the action model.

Throw Fault

This section includes the following topics:

- ♦ [“About the Throw Fault Action” on page 220](#)
- ♦ [“Adding a Throw Fault Action” on page 221](#)

About the Throw Fault Action

You use the Throw Fault action to do the following:

- ♦ Write information to an XML message on failure of an action
- ♦ Perform any number of actions before throwing the fault
- ♦ Halt execution of a component

Throw Fault is only executed when a condition that you specify is true. The message part that is written when a Throw Fault action is executed is called a Fault document, and the XML within this message is contained in a global object called ERROR.

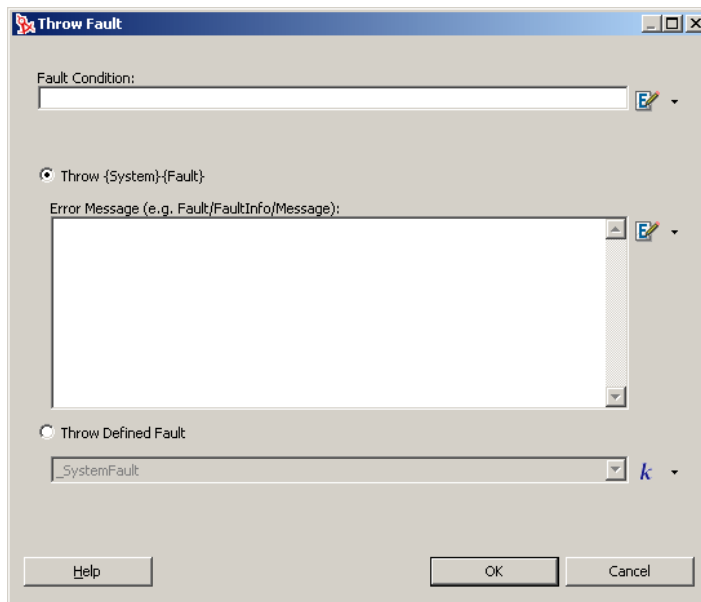
Throw Fault actions can be used in a number of ways:

- ♦ Using a Throw Fault Action by itself. You can specify a Fault Condition and an error message within the Throw Fault Action dialog. When the action is executed, the Fault Condition is evaluated. If the condition evaluates as true, the following occurs:
 - ♦ Any Before Throw actions that you specify are executed. This can be very useful as a way to leave your application in a particular state before halting execution. For example, you can send a mail message stating that the execution did not complete.
 - ♦ The contents of the error message are written to the Fault document in a node that you specify, as well as to the global object ERROR.
 - ♦ The action model execution is halted.
- ♦ Using a Throw Fault Action within a decision expression in the Decision action. You can specify a fault condition by entering it in the decision expression of a Decision action. Then put a Throw Fault statement in the true branch of the Decision action. Here you can either specify additional conditions in the Throw Fault fault condition or leave it blank and simply specify the fault document to which the fault information should be written. When the action is executed and all your conditions are true, the Throw Fault action is executed. If the fault condition in the Decision action or Throw Fault action is false, the next action in the action model is executed.
- ♦ Using a Throw Fault inside a Try /On Fault action (see [“Try/On Fault” on page 222](#)). By putting either of the above methods inside the execute branch of a Try /On Fault action, you prevent the Integration activity from halting execution and have an opportunity to respond or recover from the fault. You create your fault condition using one of the previous two methods inside the execute branch of a Try/On Fault action after other actions the output of which you want to test have worked correctly. You can specify any number of unique faults so that the Integration activity can branch into several different directions, depending on which fault occurs. When the Throw Fault action for the given fault is triggered, instead of halting execution of the component, control passes into the appropriate branch of the Try/On Fault action. Here you can specify other actions to remedy or respond to the error.

Adding a Throw Fault Action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the Throw Fault action (the new action will be inserted below the line that you selected).

- 2 Select *New Action > Advanced > Throw Fault*.



- 3 In the *Fault Condition* field, type a valid ECMAScript expression that, when true, causes the action to throw a fault.

You can also click the ECMA expression builder button and build an expression (see [Chapter 10, “Working with ECMA Expressions,” on page 249](#)).

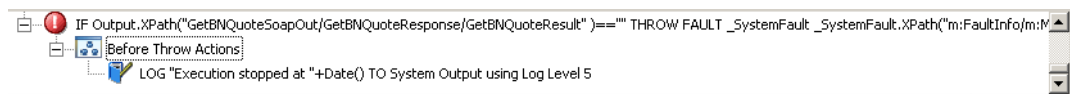
- 4 Select *Throw {System}{Fault}* to write your error message to the `_SystemFault` document.

By default, the message that you type in the *Error Message* field will be placed in the *Fault/FaultInfo/Message* node of that document. Specify a different node if desired. You can also click the ECMA expression builder button and build an expression.

- 5 Select *Throw Defined Fault* to select a fault document that is one of the message parts associated with the Integration activity.

- 6 Click *OK*.

The new action is added to the action model. Place any actions that you wish to execute before the application stops in the *Before Throw Actions* branch.



Try/On Fault

The section includes the following topics:

- ♦ [“About the Try/On Fault Action” on page 222](#)
- ♦ [“Adding a Try/On Fault Action” on page 223](#)

About the Try/On Fault Action

The Try/On Fault action executes a set of actions when a fault occurs within the Execute branch of the Try/On Fault action. Any number of defined faults can be specified within the Execute branch.

You can use the Try/On Fault action to trap anticipated errors and run other actions to remedy or report on the fault. For instance, you can use Try/On Fault to respond to an XML Interchange action that fails to find a file.

When you add a Try/On Fault action, several lines are added to the action model:

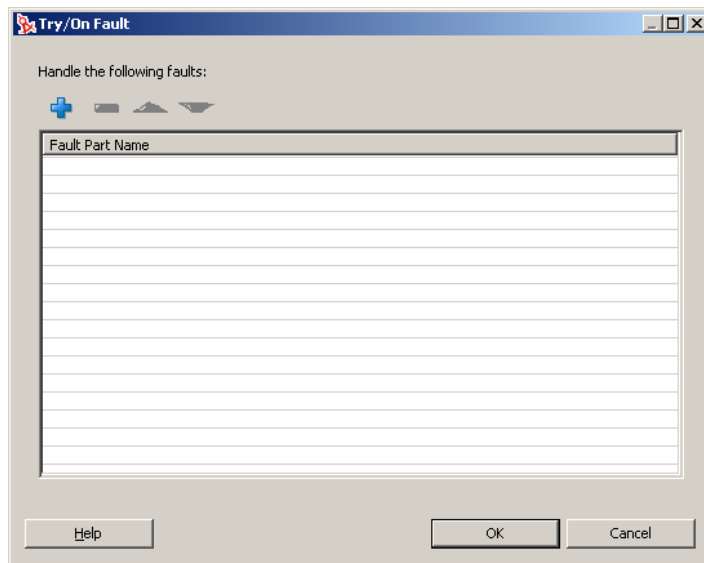
- ♦ The beginning of the Try action
- ♦ The *Execute* branch
- ♦ A branch for each Fault that you specified
- ♦ An *All other Faults* branch

When you are aware of potential faults an action can produce, you put those actions in the Execute branch. You then put error handling actions under each On Fault branch to handle unique situations. If a fault does occur, the actions in the On Fault branch execute.

Following the example given previously, if you anticipate a fault with the XML Interchange action, you put the action under the Execute branch. In one On Fault branch, you might add another XML Interchange action that attempts to read the file from an alternate location. In another On Fault branch, you might add another XML Interchange action that looks for a file with a different extension.

Adding a Try/On Fault Action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the Try/On Fault action (the new action will be inserted below the line that you selected).
- 2 Select *New Action > Advanced > Try/On Fault*.



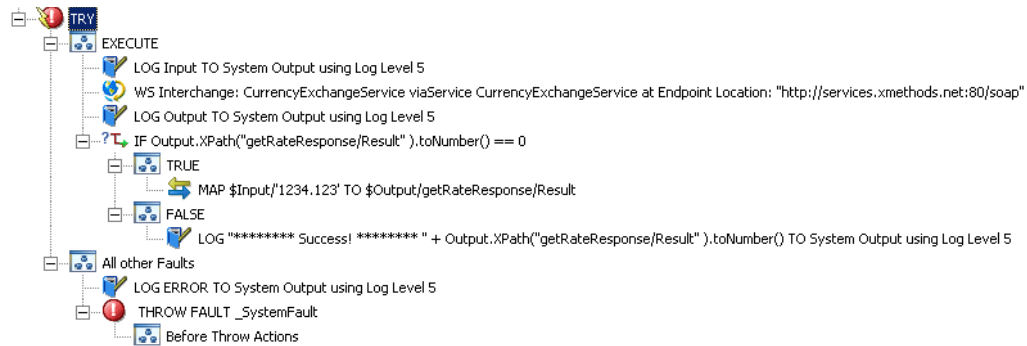
Use the + icon to add a new fault part to the *Fault Part Name* list. Use the red - icon to remove fault parts from the list. Use the up and down arrow icons to change the order of the faults.

If you don't specify a fault part, corrective actions can be placed in the default "All Other Faults" branch of the Try/On Fault action.

- 3 Click *OK*. The following appears in the Action Model Viewer: the *Try On Fault* action icon, with an Execute, one or more On Fault branches, and an All Other Faults branch.

- 4 Add any actions that might cause errors to the Execute branch.
 - 5 In the On Fault branch, add actions that resolve the errors specified in the Execute branch.
- The following illustration shows a complete Try/On Fault action in the action model.

Figure 9-10 Try/on Fault Action Example



9.5.2 Data Exchange

This section includes descriptions of the following actions:

- ♦ [“WS Interchange” on page 224](#)
- ♦ [“XML Interchange” on page 226](#)

WS Interchange

This section includes the following topics:

- ♦ [“About the WS Interchange Action” on page 224](#)
- ♦ [“Adding a WS Interchange Action” on page 224](#)

About the WS Interchange Action

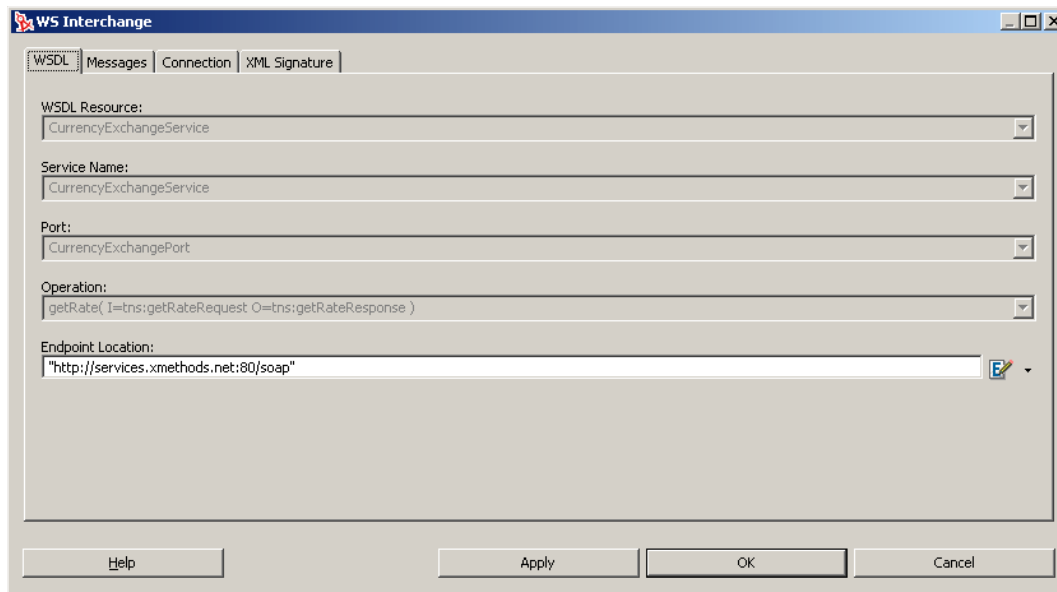
The WS (Web Service) Interchange action is the most important action in the Integration activity and allows the Integration activity to invoke a Web service according to calling conventions specified in a WSDL file. The Integration activity creates a WS Interchange action automatically when it creates the action model.

In most cases there will be no need to add another WS Interchange action to the action model. However, there might be situations in which you need to do so. The following procedure describes how to add a WS Interchange action.

Adding a WS Interchange Action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the WS Interchange action (the new action will be inserted below the line that you selected).

- 2 Select *New Action > Data Exchange > WS Interchange*.



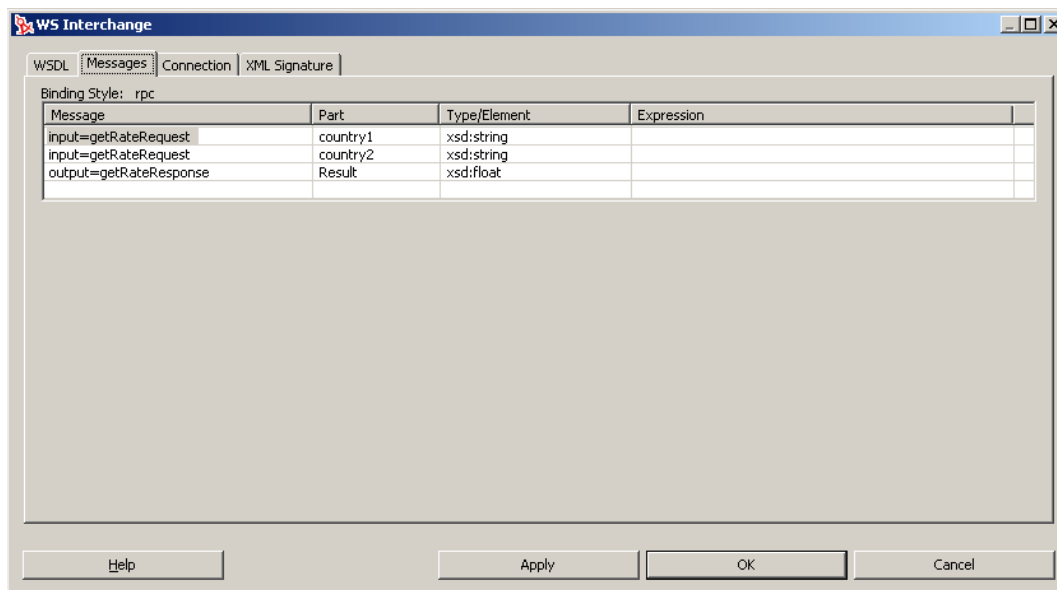
The WS Interchange dialog box is shown with the 'WSDL' tab selected. It contains several fields for configuring a web service:

- WSDL Resource:** CurrencyExchangeService
- Service Name:** CurrencyExchangeService
- Port:** CurrencyExchangePort
- Operation:** getRate(I=tns:getRateRequest O=tns:getRateResponse)
- Endpoint Location:** "http://services.xmethods.net:80/soap"

Buttons at the bottom include Help, Apply, OK, and Cancel.

The *WSDL Resource*, *Service Name*, *Port*, and *Operation* fields are filled in automatically based on the information in the WSDL specified for the Integration activity.

- 3 If desired, modify the information in the *Endpoint Location* field (usually a URL pointing at a servlet) for the Web service that you wish to use, wrapped in quotation marks. Alternatively, enter an ECMAScript expression that will evaluate to an Endpoint Location at runtime.
- 4 Click the *Messages* tab.



The WS Interchange dialog box is shown with the 'Messages' tab selected. It displays a table for message binding information:

Message	Part	Type/Element	Expression
input=getRateRequest	country1	xsd:string	
input=getRateRequest	country2	xsd:string	
output=getRateResponse	Result	xsd:float	

Buttons at the bottom include Help, Apply, OK, and Cancel.

The *Message*, *Part*, and *Type/Element* fields are filled in automatically based on the information in the WSDL specified for the Integration activity.

- 5 If desired, click the *Expression* column for a message, then use the ECMA expression builder to create an ECMAScript expression that describes the source and target for the message.

Usually, this will be an expression that specifies an XPath location in an Input part or Output part.

6 Click the *Connection* tab.

You use this tab to specify connection parameters for HTTP servers that require authentication.

7 Type a user ID to use for the connection in the *User ID* field, and a password for the user in the *Password* field. The user ID and password are not actually submitted during the establishment of a connection. They are simply defined here. The password is encrypted. You will have access to UserID and Password variables in ECMAScript, allowing you to map the user ID and password as values into the screen. This way, no one ever sees the passwords.

8 If the connection requires a client certificate, choose a client certificate by clicking on the browse button in the *Client Certificate* field and selecting the certificate file you want to use for this connection.

9 If the connection requires a client private key, choose a client private key by clicking on the browse button in the *Client Private Key* field and selecting the client private key file.

10 Type the password for the client private key in the *Private Key Password* field.

11 Specify a connection timeout value, in seconds, in the *Connection Timeout* field.

12 Click *Apply* to test the WS Interchange action in real time, or click *OK* to close.

XML Interchange

This section includes the following topics:

- ♦ [“About the XML Interchange Action” on page 226](#)
- ♦ [“Adding an XML Interchange Action” on page 227](#)

About the XML Interchange Action

The XML Interchange action reads external XML documents into a DOM and writes data from the DOM as XML files. There are four types of XML Interchange actions:

- ♦ GET
- ♦ PUT
- ♦ POST
- ♦ POST with Response

When using the Get interchange, fill in the *Interchange URL Expression* field with a URL that points to the XML document that you want to bring into the Integration activity. In the *Response Part* field, you select the DOM (Input, Output, _SystemFault, or Project) that is to receive the XML.

When using the Put interchange, enter a URL that points to the location to which you want to write the XML document in the *Interchange URL Expression*. In the *Request Part* field, you select the name of the DOM from which you want to send data as XML.

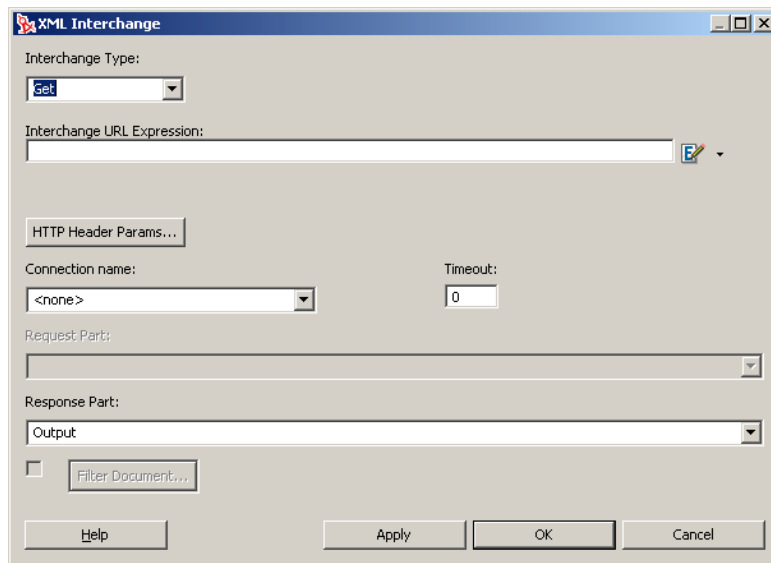
When using the Post interchange, enter a URL that points to the location to which you want to write the XML document in the *Interchange URL Expression* field. In the *Request Part* field, you select the name of the DOM from which you want to send data as XML.

When using the Post with Response interchange, you supply the same parameters as for Post, with one additional parameter. You must also specify a Response Part DOM to receive the Response

XML document from the Post with Response interchange. The difference between the two interchanges is that Post with Response expects a response XML object back from the origin server.

Adding an XML Interchange Action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the XML Interchange action (the new action will be inserted below the line that you selected).
- 2 Select *New Action > Data Exchange > XML Interchange*.



- 3 Select an *Interchange Type* (Get, Put, Post, or Post with Response).
- 4 In the *Interchange URL Expression* field, type an expression that defines a fully qualified URL for an XML document, using any of the following supported protocols:
 - ♦ file
 - ♦ ftp
 - ♦ http
 - ♦ https

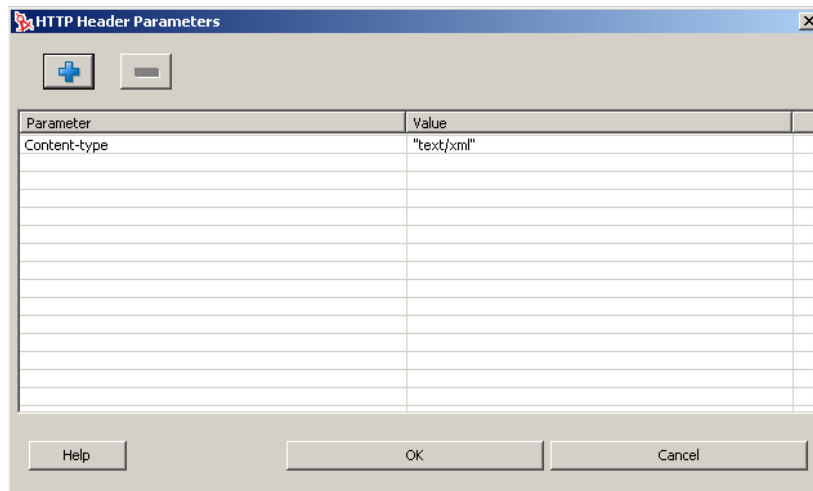
Depending on the *Interchange Type* selected, this URL is the source or the destination of the XML file for the XML Interchange action. For example:

file:///g:/xmldata/invoicebatch1.xml

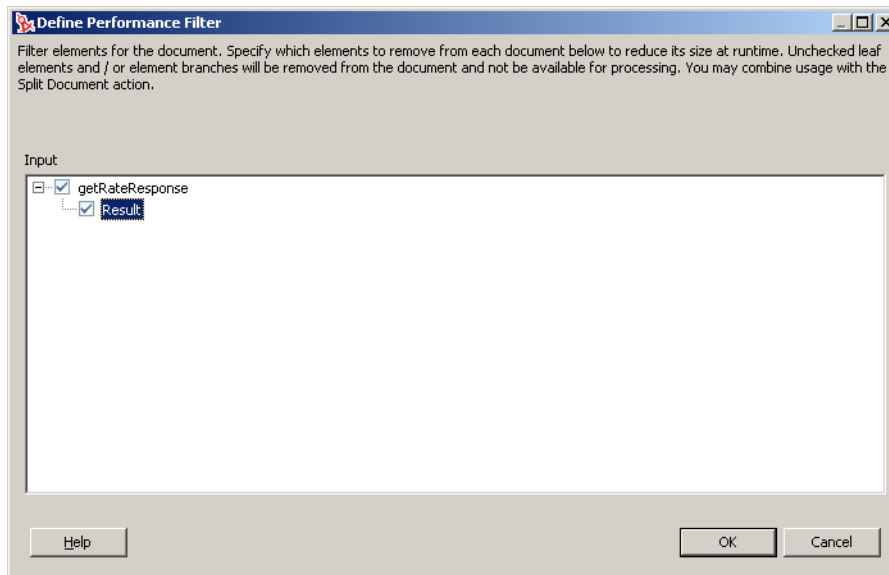
ftp://accounting:password@123.456.789.987:21/invoices/inv1.xml

Since this is an ECMAScript expression, the URL string must be enclosed in quotation marks.

- 5 If you need to specify HTTP header parameters, click *HTTP Header Parameters*.



- 6 Click the plus (+) icon to add new header parameters, then type a *Parameter* name and a corresponding *Value*. Common HTTP header parameters include “Content-Type,” “Content-Length,” and “Keep-Alive.” You can add any number of Parameter-Value pairs.
- 7 Click *OK* to return to the *XML Interchange* dialog box.
- 8 In the *Request Part* field (which is enabled if the *Interchange Type* is Put, Post, or Post with Response), select the name of the DOM from which you want to send data as XML.
- 9 In the *Response Part* field (which is enabled if the *Interchange Type* is Get or Post with Response), select the name of the DOM tree that will receive the XML.
- 10 If you would like to filter the incoming XML document to improve performance, select the check box next to the *Filter Document* button, then select the *Filter Document* button.



The document displayed is the document selected in the *Response Part* in the *XML Interchange* dialog box.

You use this dialog box to specify the individual nodes to retain (rather than discard) from the incoming XML document in real time to improve performance and reduce RAM overhead.

- 11 Check the nodes that you want to keep in the document.

Unchecked nodes are discarded before parsing the DOM.

- 12 When you have selected nodes that you want to keep, click *OK* to return to the *XML Interchange* dialog box.

- 13 Click *OK*.

Alternatively, you can click *Apply* to see the affect of the XML Interchange action without closing the dialog box. This allows you to make repetitive edits to an XML Interchange action and quickly see the results.

9.5.3 Repeat

This section includes descriptions of the following actions:

- ♦ “Break” on page 229
- ♦ “Continue” on page 230
- ♦ “Declare Group” on page 230
- ♦ “Repeat for Element” on page 232
- ♦ “Repeat for Group” on page 233
- ♦ “Repeat While” on page 235

Break

This section includes the following topics:

- ♦ “About the Break Action” on page 229
- ♦ “Adding a Break Action” on page 229

About the Break Action

The Break Action stops the execution of a Repeat for Element, Repeat for Group, or Repeat While loop. The action model continues execution with the next action outside the loop.

The use of Break is appropriate when, for example, you are using a loop to search a node list for one particular item. When the target item is found, there is no need to continue iterating. A Break can be used to terminate the loop immediately.

NOTE: A Break action is usually used in one branch of a Decision action (within a loop). You place the Break action in either the True or False branch of the Decision action.

Adding a Break Action

- 1 Within a Repeat action that you want to modify to include a Break action, select a position inside the loop where you want to place the Break action.

Generally, this will be in one leg or the other of a Decision action.

- 2 Select *New Action > Repeat > Break*.

The Break action is inserted into the action model.

Continue

This section includes the following topics:

- ♦ [“About the Continue Action” on page 230](#)
- ♦ [“Adding a Continue Action” on page 230](#)

About the Continue Action

The Continue action causes execution of the current iteration of a Repeat for Element, Repeat for Group, or Repeat While loop to stop and execution to begin at the top of the loop, with the next iteration. The Continue action provides a way to pass over downstream actions inside the loop while allowing the loop to continue on to the next iteration.

A Continue action is appropriate in a situation where, for example, one item in a list should be skipped for some reason, yet execution of the loop must continue.

NOTE: A Continue action usually occurs in one branch of a Decision action within a loop. You place the Continue action in either the True or False branch of the Decision action, as appropriate.

Adding a Continue Action

- 1 Within a Repeat action that you want to modify to include a Continue action, select a position inside the Loop actions where you want to place the Continue action.

This is usually inside one fork or the other of a Decision action.

- 2 From the Action menu, select *New Action > Repeat > Continue*.
A Continue action appears in the action model.

Declare Group

This section includes the following topics:

- ♦ [“About the Declare Group Action” on page 230](#)
- ♦ [“Adding a Declare Group Action” on page 231](#)

About the Declare Group Action

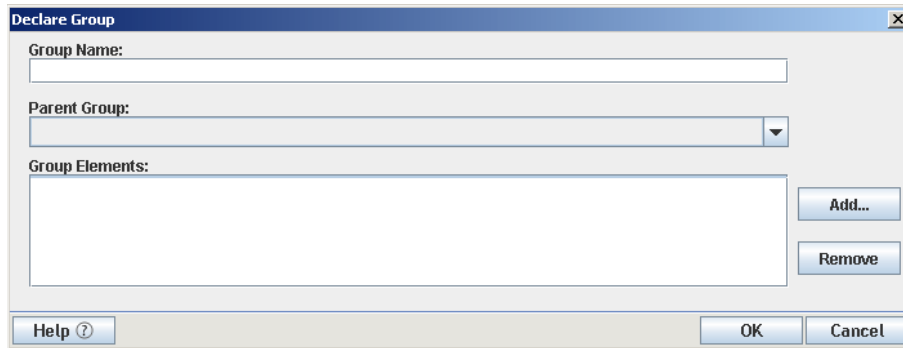
You use the Declare Group action to create two special lists, each in reference to a DOM. These group lists can then be used as the basis for a loop in the Repeat for Group action. To create the lists, you supply a Group Name and specify an XPath. The Integration activity then creates the lists as follows:

- ♦ A Group list is created that contains one entry for each unique value found in all the elements that match the XPath. The Group list is referred to by the Group Name that you supply.
- ♦ A Detail list is created for each unique entry in the Group list that contains as many entries as there are members in the Group. The Detail list is referred to by the group name that you supply, post-fixed with the label “(Detail).”

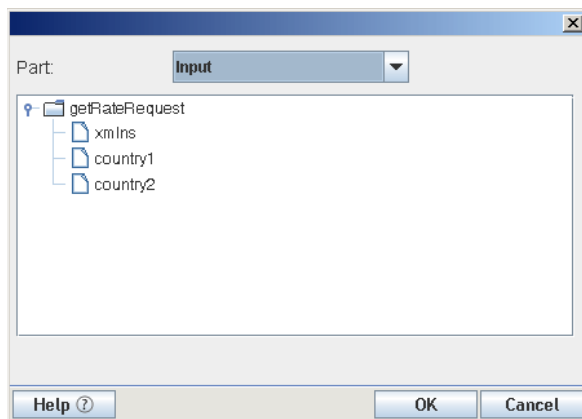
Using Groups allows you to select a repeating element in your Input DOM and create fewer elements based on the unique values across all siblings of that repeating element. Instead of having multiple elements, you have one element for each unique element value in your Output DOM.

Adding a Declare Group Action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the Declare Group action (the new action will be inserted below the line that you selected).
- 2 Select *New Action > Repeat > Declare Group*.



- 3 Type a name for the group in the *Group Name* field.
- 4 If you would like to create multiple group levels, select a group from the *Parent Group* list, which lists groups that you have previously defined.
- 5 Click Add. The *Add Element* dialog box is displayed.



- 6 Select a part name and an element.
- 7 Click *OK*.
- 8 Repeat **Step 5** through **Step 7** to add more elements to the group.
- 9 When you have all the elements that you want in the group, click *OK*.

Repeat for Element

This section includes the following topics:

- ♦ [“About the Repeat for Element Action” on page 232](#)
- ♦ [“Adding a Repeat for Element Action” on page 232](#)

About the Repeat for Element Action

The Repeat action creates looping structures within an action model. Loops give you the ability to repeat a set of one or more actions. There are three types of loops: Repeat for Element, Repeat for Group, and Repeat While.

XML allows multiple instances of an element in a document (analogous to multiple records in a database table). The number of instances can vary from document to document and is defined in the document schema (DTD or XML Schema). For example, you might receive an XML document containing line items for an invoice on a daily basis. Each day the XML document has a different number of line items. Not knowing how many instances of “line item” are in the XML document poses a problem if you want to transfer these item numbers from the input XML document to an output XML document programatically. The Repeat for Element action solves this problem.

The Repeat for Element action allows you to mark an element that occurs multiple times. The action then sets up a processing loop that executes one or more actions for each instance of the marked element until no more instances exist. In the previous example, the processing loop would contain a single Map action to transfer the line item number and this action would be repeated until all line items had been mapped.

The Repeat for Element action also uses the concept of an alias. An alias performs two functions. It is an alternate name or shorthand for the marked repeating element, which saves you the work of re-specifying long XPath expressions. In some cases, the repeating element might be several levels down in the document hierarchy. When you create Map actions in the Repeat loop that transfers child elements of the marked element, using the alias is quicker than re-typing a long XPath expression. An alias is also an indicator to Map actions within the Repeat loop to use the next instance of the repeating element each time the loop processes. A Map action within a Repeat for Element loop that does not use the alias always refers to the first instance of the element in the source message.

The Repeat for Element action allows you to process more than one action within the loop. In the simplest case, the repeat loop might only contain one Map action that transfers the value of the current element instance from the input Part to the output Part. You can also define multiple actions in the processing loop. For example, a Map action to transfer the current value, and a Log action that writes to a file, creating an audit of each transfer.

Adding a Repeat for Element Action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the Repeat for Element action (the new action will be inserted below the line that you selected).

2 Select *New Action > Repeat > Repeat for Element*.

The 'Repeat for Element' dialog box is shown. It has a title bar with the text 'Repeat for Element'. The dialog is divided into two main sections: 'Source' and 'Target'.
In the 'Source' section:
- There is an 'Alias:' label followed by a text input field.
- Below that is a 'Representing:' label with two radio buttons: 'XPath:' (which is selected) and 'Expression:'.
- Below the radio buttons is a large text input field for the XPath or Expression.
In the 'Target' section:
- There is an 'Alias:' label followed by a text input field.
- To the right of the 'Alias' field is a checkbox labeled 'Always create new output elements'.
- Below that is a 'Representing:' label with two radio buttons: 'XPath:' (which is selected) and 'Expression:'.
- Below the radio buttons is a large text input field for the XPath or Expression.
At the bottom of the dialog are three buttons: 'Help', 'OK', and 'Cancel'.

3 Specify the *Source* information.

3a Type an alias name in the Source *Alias* field.

A good naming convention for an alias is to use the element name with a prefix indicating source or target and the type of repeat action, such as “SILineitem.”

3b Type an XPath expression, or click the ECMA expression builder button and build an XPath expression for the repeating element.

4 Specify the *Target* information.

4a Type an alias name in the Target *Alias* field.

4b Check *Always create new output elements* if you have repeating actions that should add new elements rather than updating existing elements.

4c Enter an XPath expression, or click the ECMA expression builder button and build an XPath expression for the repeating element.

5 Click *OK*. The Repeat for Element loop is added to the action model.

6 Click *Loop Actions* in the action model to begin adding actions to be performed within the loop.

Repeat for Group

This section includes the following topics:

- ♦ “About the Repeat for Group Action” on page 234
- ♦ “Adding a Repeat for Group Action” on page 234

About the Repeat for Group Action

The format of an XML document that you receive is not always the format that meets the requirements of your business process. A Repeat for Group action allows you to restructure data and establish a framework to calculate aggregates on your data. Grouping allows you to select a repeating element in your input part and create fewer elements based on the unique values across all siblings of that repeating element.

The Repeat for Group action sets up a processing loop based on one of two lists created by the Declare Group action. The loop executes as many times as there are entries in the list you use (either the Group list or Detail list). By combining a Repeat for Group with Map commands, you can create a new XML document that has different structure and data from the original.

Similar to the Repeat for Element action, a Repeat for Group action also uses the concept of an alias. The values for source group used in the Repeat for Group dialog are the list names created by the Declare Group action. The list names perform two functions. They are an alternate name or short-hand for the XPath source of any Map actions within the loop. This saves you the work of re-specifying long XPath expressions. The group list name, when used in place of a DOM name in a Map action source, is also an indicator to the Map action within the Repeat loop to use the next instance in the group list each time the loop processes. A Map action within a Repeat for Group loop that does not use the group name always refers to the first instance of the element in the source part.

The target aliases created in the Repeat for Group action also serve two functions. They are an alternate name or short-hand for the XPath target of any Map actions within the loop. This saves you the work of re-specifying long XPath expressions. The target alias, when used in place of a part name, is also an indicator to Map actions within the Repeat loop to create a new instance of the Source in the target message part. A Map action within a Repeat for Group loop that does not use a target alias always overwrites the first instance created in the target message part with subsequent instances from the source group list.

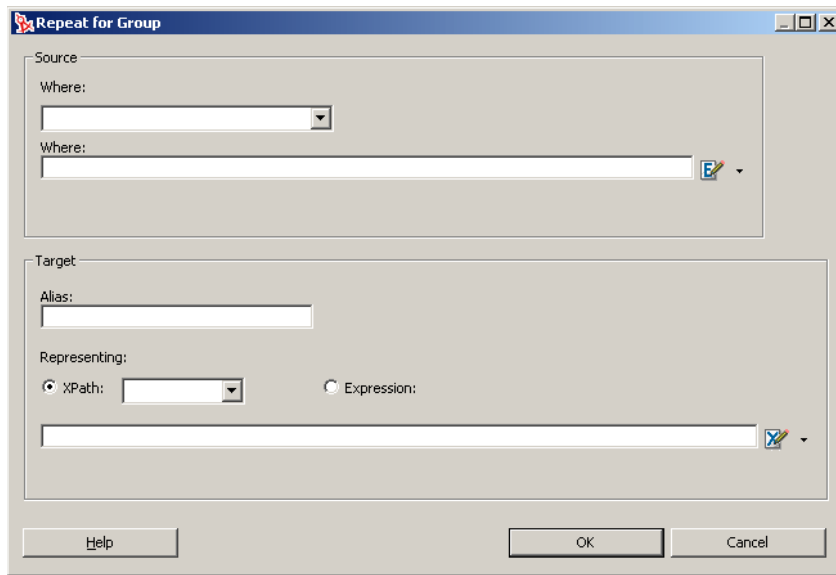
Creating a Repeat for Group action consists of three tasks:

- ♦ Create a Declare Group action to create the group lists.
- ♦ Create a Repeat for Group action specifying which group list to use.
- ♦ Create Map actions inside the loop.

Adding a Repeat for Group Action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the Repeat for Group action (the new action will be inserted below the line that you selected).

- 2 Select *Action > New Action > Repeat > Repeat for Group*.



- 3 In the Source section, select a Group name from the *Where* list on which to base the Repeat for Group action loop.
- 4 Optionally, type a Where clause in the *Where* field to filter the group list, or click the ECMA expression builder icon and create a Where expression.
- 5 If you know the alias for the Target element, type the name in the *Alias* field.
- 6 If you do not know the alias, select either the *XPath* button and select an element from the list, or select the *Expression* button and type an expression (or click the ECMA expression builder button and build an expression).
- 7 Click *OK*.

Repeat While

This section includes the following topics:

- ♦ [“About the Repeat While Action” on page 235](#)
- ♦ [“Adding a Repeat While Action” on page 236](#)

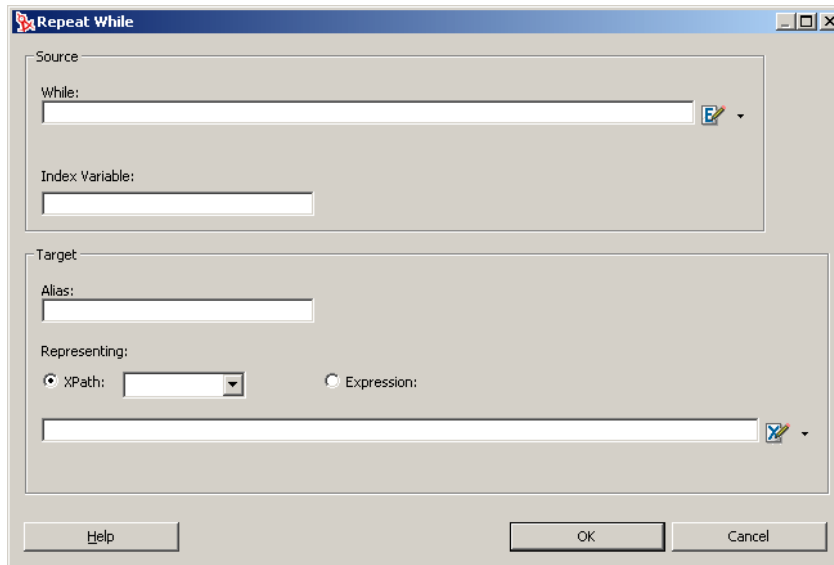
About the Repeat While Action

The Repeat While action repeats one or more actions as long as a condition that you specify remains true. The target alias that you create in the Repeat While action serves two functions. It is an alternate name or short-hand for the XPath target of any Map actions within the loop. This saves you the work of re-specifying long XPath expressions. The target alias, when used in place of a DOM name in a Map action, is also an indicator to Map actions within the Repeat loop to create a new instance of the source in the target DOM. A Map action within a Repeat for Group loop that does not use a target alias always overwrites the first instance created in the target DOM with subsequent instances from the source.

NOTE: Unlike the Repeat for Element and Repeat for Group, the Repeat While does not have to be based on data in a DOM tree. The loop can operate independently of data in the DOM tree.

Adding a Repeat While Action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the Repeat While action (the new action will be inserted below the line that you selected).
- 2 Select *New Action > Repeat > Repeat While*.

The image shows a 'Repeat While' dialog box with a title bar. It is divided into two main sections: 'Source' and 'Target'. The 'Source' section contains a 'While:' text field with an ECMA expression builder icon to its right, and an 'Index Variable:' text field below it. The 'Target' section contains an 'Alias:' text field, followed by a 'Representing:' section with two radio buttons: 'XPath:' (which is selected) and 'Expression:'. Below these is a large text field for the XPath or expression, also with an ECMA expression builder icon to its right. At the bottom of the dialog are three buttons: 'Help', 'OK', and 'Cancel'.

- 3 In the *While* field, type an expression that tests the While loop, or click the ECMA expression builder button and build an expression.
- 4 In the *Index Variable* field, type a name for a variable that will keep track of the condition of the loop.
- 5 If you know the alias for the Target element, type the name in the *Alias* field.
- 6 If you do not know the alias, select either the *XPath* button and select an element from the list, or select the *Expression* button and type an expression (or click the ECMA expression builder button and build an expression).
- 7 Click *OK*.

9.5.4 Comment

This section includes the following topics:

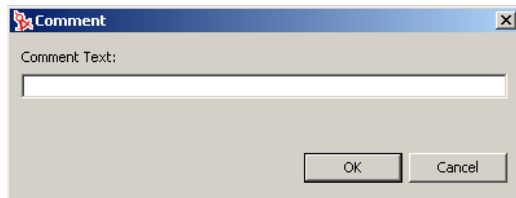
- ♦ [“About the Comment action” on page 236](#)
- ♦ [“Adding a Comment action” on page 237](#)

About the Comment action

You can use the Comment action to document the action model and clarify the processing that takes place. You can add comments anywhere within an action model. Comments perform no processing of their own.

Adding a Comment action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the comment (the new action will be inserted below the line that you selected).
- 2 Select *New Action > Comment*.



- 3 Type your comment.
- 4 Click *OK*.

9.5.5 Decision

This section includes the following topics:

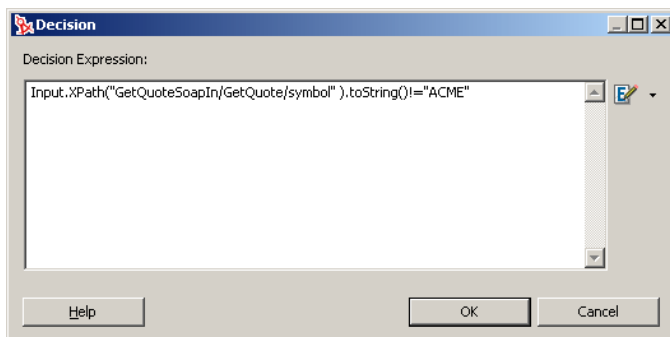
- ♦ [“About the Decision action” on page 237](#)
- ♦ [“Adding a Decision action” on page 237](#)

About the Decision action

The Decision action creates an *if* . . . *then* construct between actions or a group of actions. You use a Decision action to select a branch, based on a condition that you supply. The condition must use an ECMAScript comparison operator, such as `=`, `<`, `>`, `!`, `>=`, `<=`, `(&)`, `OR (||)`, or `<>`. The expression must resolve to a Boolean true or false statement.

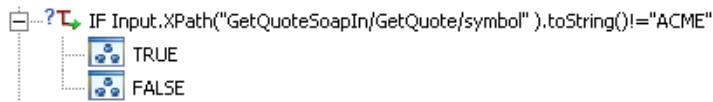
Adding a Decision action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the Decision action (the new action will be inserted below the line that you selected).
- 2 Select *New Action > Decision*.



- 3 Type an ECMAScript expression, or click the ECMA expression builder button and create a Decision script that will evaluate to *true* or *false* at runtime.
- 4 Click *OK*.

A Decision action similar to the following is displayed.



- 5 In the action model pane, select the *TRUE* branch.
- 6 Add one or more actions that will execute if the expression is true.
- 7 Select the *FALSE* branch.
- 8 Add one or more actions that will execute if the expression is false.

You can nest other Decision actions inside the TRUE or FALSE branches of the Decision action.

9.5.6 Function

- ♦ [“About the Function Action” on page 238](#)
- ♦ [“Adding a Function Action” on page 238](#)

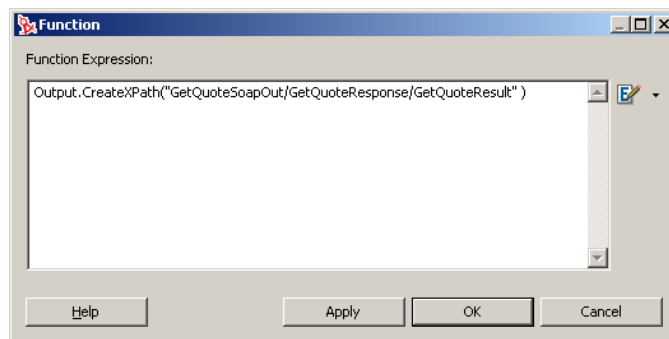
About the Function Action

The Function action executes an ECMAScript function. To manipulate a DOM element, the script that you call in the Function action must reference a fully qualified DOM element name in the current Integration activity.

Custom Script functions that you create and add to an action model can act upon any XML tree element. For example, you can create a function that changes the format of a date element. You can create a function that performs a math function on the contents of an element. You can also perform file system, database, or URL functions that have no interaction with a message part.

Adding a Function Action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the Function action (the new action will be inserted below the line that you selected).
- 2 Select *New Action > Function*.



- 3 Type the function in the *Function Expression* field or click the ECMA expression builder button to build an ECMAScript expression.
- 4 Click *OK*.

9.5.7 Log

- ♦ “About the Log action” on page 239
- ♦ “Adding a Log action” on page 239

About the Log action

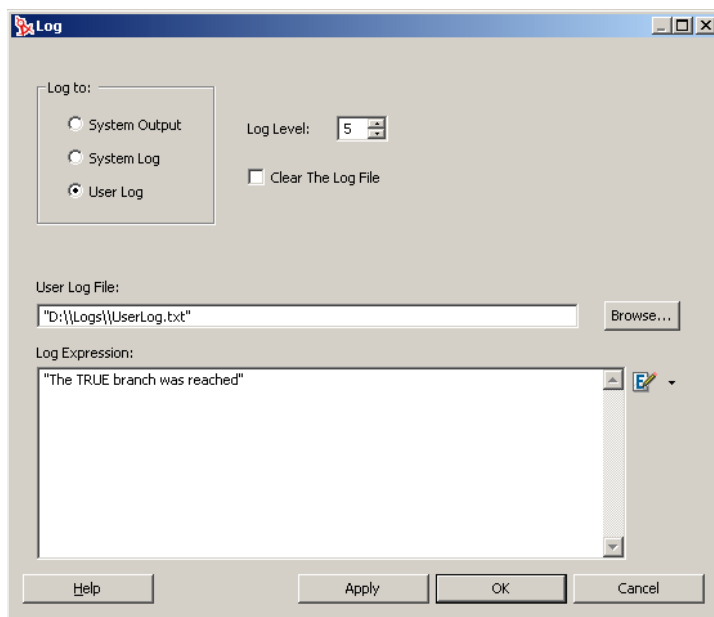
Log actions provide customizable reporting capabilities (design-time as well as runtime) for Integration activities. You can specify log level settings to control the degree of reporting.

Some Log usage examples include the following:

- ♦ Writing certain error information to the operator console when a Try On Fault condition is reached.
- ♦ Using ECMAScript expressions to aid in debugging by logging information about variables or DOM contents, the values of which are known only at runtime.
- ♦ Capturing specific information from each cycle of a Repeat for Element loop.

Adding a Log action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the Log action (the new action will be inserted below the line that you selected).
- 2 Select *New Action > Log*.



- 3 Select the type of log that you would like to produce from the *Log to* group.

The Log action writes information to locations specified in the action. There are three locations for log output: System Output, System Log, and User Log.

Log Location	Description
<i>System Output</i>	Select System Output to write messages that you specify in the <i>Log Expression</i> field to the operating system console at design time, or the application server console at runtime. NOTE: To view messages on the operating system console, start Designer using the Eclipse <code>-debug</code> and <code>-consoleLog</code> startup parameters.
<i>System Log</i>	Select System Log to write messages that you specify in the <i>Log Expression</i> field to the application server log file.
<i>User Log</i>	Select User Log to write messages that you specify in the <i>Log Expression</i> field to a file that you specify in the <i>User Log File</i> field.

- 4 Use *Log Level* to select a priority level (1 to 10) for this Log action.

The default priority level is 5. You should assign a number from 5 to 10 to messages that you want to appear in the log file. The priority you assign here will be compared to the threshold number (which is set to 5 internally and cannot be changed). If the priority is equal to or greater than the threshold, the message is logged; otherwise it is not.

- 5 Check *Clear the Log File* if you want the data in the log file to be cleared each time the component is executed.

- 6 If *User Log* is selected in the *Log to* group, type the path to the log file in the *User Log File* field, or use *Browse* to specify a log file.

If you specify a file that doesn't exist, the file will be created. On Windows* systems, if you type the path, you must add an extra backslash character wherever a backslash character occurs in a path (for example, `C:\Windows` becomes `C:\\Windows`).

- 7 Create the message that you want to record to the log in the *Log Expression* field.

You can type a message in the field or use the ECMA expression builder to build an expression.

9.5.8 Map

This section includes the following topics:

- ♦ [“About the Map Action” on page 241](#)
- ♦ [“Adding a Map Action” on page 242](#)
- ♦ [“Advanced Mapping Options” on page 243](#)
- ♦ [“Transforming Elements with the Content Editor” on page 245](#)

About the Map Action

The Map action performs DOM-node input and output mapping. It can transfer and transform data from one document context to another document context. A context has two parts. The first part usually identifies a DOM and the second part identifies a location within the DOM. The basic document context in an Integration activity is expressed as a DOM name combined with an element location identified through an XPath expression. The DOM name is usually Input, Output, _System Fault, or Project. The XPath expression identifying a location in a DOM has the path elements delimited by “/”.

NOTE: A context in an Integration activity can also be a Group name that itself is simply an alias or short-hand for an XPath expression.

Default Mapping Behavior

When you use the Map action to map elements and attributes within XML documents, certain default behaviors occur. The following table lists those default behaviors.

Table 9-5 *Default Mapping Behavior*

Map Type	Default Behavior
Leaf Element to Leaf Element	Transfers the element data only.
Leaf Element to Branch Element	Transfers the element data only.
Branch Element to Leaf Element	Transfers the entire branch, including all child elements and attribute data under the branch.
Branch Element to Branch Element	Transfers the entire branch, including all child elements and attribute data under the branch after removing the target's current branch.
A particular Leaf Element in a list of Leaf Elements, to Element	Transfers the element data from the selected leaf (or element instance) to the target element.
Attribute to Attribute	Transfers the attribute data only.
Element to Attribute	Transfers element data to attribute data.
Attribute to Element	Transfers the attribute data only.

Many of these behaviors can be altered, on an action-by-action basis, through the use of the Advanced mapping dialog (see [“Advanced Mapping Options” on page 243](#)).

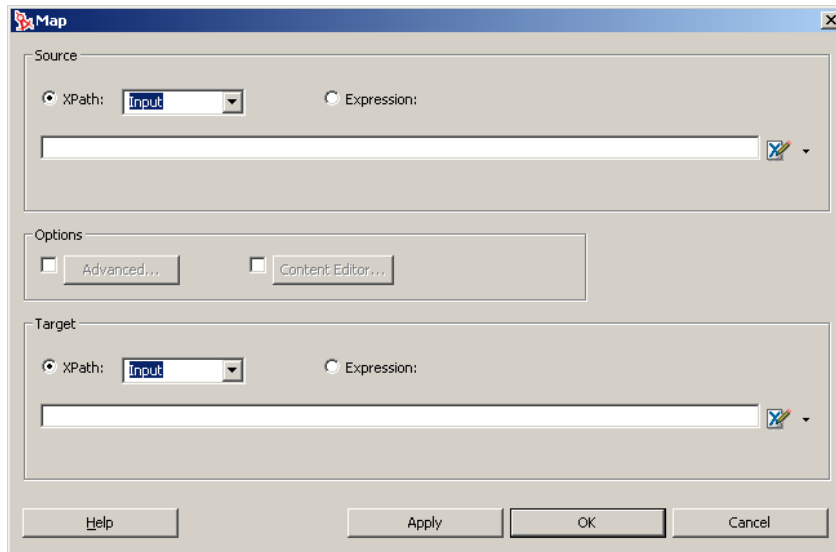
Leaf Elements that Contain Markup

A problem can occur when an element is populated at runtime by a Java or ECMAScript operation. The element might receive data that contains markup (strings with illegal characters, such as < and >). If the Integration activity were to map the contents of such an element to a node in the Output DOM, the output document would be malformed. The Integration activity resolves this issue by mapping any data that contains markup to a new CDATA section in the target document.

NOTE: When markup is entered at design time, the behavior is different. If you type markup into a node, and you examine the raw XML in *Source* view, you'll see that markup characters typed into a node are converted to entities. For example, a "<" character is converted to `<`.

Adding a Map Action

- 1 Right-click the line in the action model that is one line above the position in which you want to place the Map action (the new action will be inserted below the line that you selected).
- 2 Select *New Action > Map*.



- 3 In the *Source* section, select *XPath*.
- 4 Select a part (Input, Output, _SystemFault, or Project) from the list, then type the appropriate XPath expression, or use the ECMA expression builder to locate the element that you want.
Together, the part name and XPath specify the Source context for the Map action.
- 5 Repeat [Step 3](#) and [Step 4](#) for the *Target* section.
- 6 If you want more control over mapping, select the *Advanced* (see [“Advanced Mapping Options” on page 243](#)) or *Content Editor* (see [“Transforming Elements with the Content Editor” on page 245](#)) options.

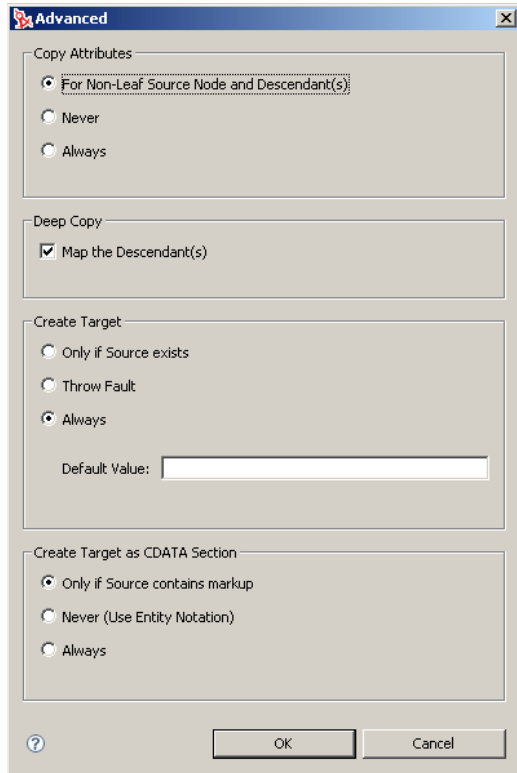
You can click *Apply* to see the effect of the Map action without closing the dialog box. This allows you to make repetitive edits to a Map action and quickly see the results.

- 7 Click *OK*.

Advanced Mapping Options

When you select the *Advanced* option in the *Map* dialog box, the *Advanced* dialog box is displayed. Options that you set in the Advanced dialog box only affect the current Map action.

Figure 9-11 *Advanced*



The options in this dialog box give you fine control over how input part nodes are mapped to the output part.

This section includes the following topics:

- ♦ “Copy Attributes” on page 243
- ♦ “Deep Copy” on page 244
- ♦ “Create Target” on page 244
- ♦ “Create Target as CDATA Section” on page 245

Copy Attributes

You use *Copy Attributes* to specify how attributes are mapped. *Copy Attributes* has the following options:

Table 9-6 *Copy Attributes Options*

Option	Description
<i>For Non-leaf Root Nodes and Dependents</i>	Specifies that when a non-terminal (non-leaf) element is mapped to output, the element (minus its attributes) and its children are mapped to output. Attribute data for the children is included, but not for the original (parent) element.
<i>Never</i>	Specifies that no attribute data (whether for parent or leaf nodes) will be carried over during mapping.
<i>Always</i>	Specifies that all attribute data, for all nodes, will be mapped to output.

Deep Copy

The default Integration activity behavior is to map whole branches at a time (the target node plus all of its children). This is referred to as a *deep copy*. In some cases, you might want to turn off this behavior so that you can copy just the parent element without its children. Deselect *Map the Dependents* if you want to disable deep copy.

Create Target

You use *Create Target* to create the destination node that you specified in the *Target* group in the Map action (see [“Adding a Map Action” on page 242](#)), based on whether or not the source node is present in the source DOM. By default the Integration activity always creates the target, whether or not the runtime source DOM contains the nodes specified in the Source XPath for mapping.

For example, in the Map action, you might have specified a Source XPath that looks like
`$Input/Root/MySourceElement`

In the Target XPath, you may have specified
`$Output/Root/MyParentNode/SomeOtherElement`

If the incoming Input document does not have a node corresponding to `Root/MySourceElement`, the Integration activity will by default create an empty `Root/MyParentNode/SomeOtherElement` node in the output DOM. In some cases, this might not be what you want. Using *Create Target* mapping, you can change the default behavior.

Table 9-7 *Create Target*

Option	Description
<i>Only if Source exists</i>	Specifies that the Map Action will be skipped (no target nodes are created in the output DOM) if the node specified in the Source XPath doesn't exist in the input message.
<i>Raise Error</i>	Specifies that if the input document doesn't contain the node specified in the Source XPath, it will be considered an error at runtime. You should plan accordingly by wrapping your Map action in a Try/OnError block so that you can handle the error.
<i>Always</i>	Specifies that the target node should always be created (the default behavior). When <i>Always</i> is selected, you can use the <i>Default Value</i> field to specify a default data value for the target element.

Create Target as CDATA Section

You use *Create Target as CDATA* to control the way element data gets mapped into CDATA sections.

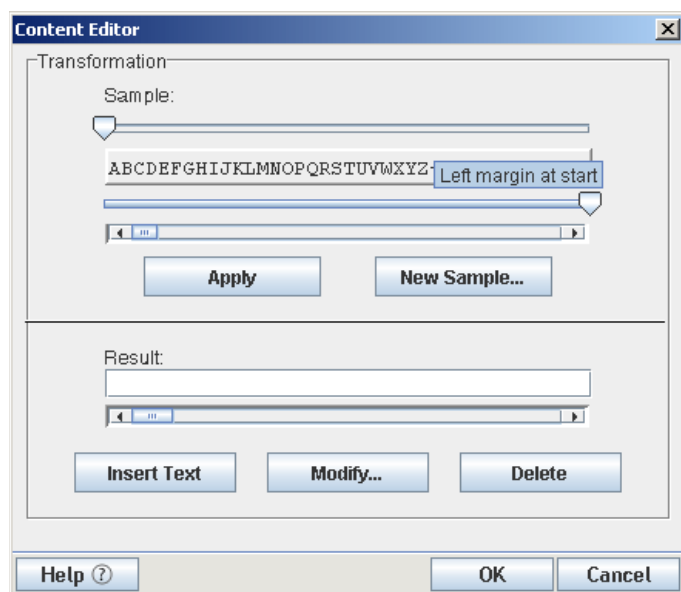
Table 9-8 *Create Target as CDATA Section*

Option	Description
<i>Only if source contains markup</i>	Specifies that if the source data contains XML, HTML, or other types of markup in which illegal (in this context) characters are used, the data will be placed, unmodified, in a CDATA section in the target DOM. This is the default behavior.
<i>Never</i>	Specifies that source data will not be wrapped in a CDATA section for output. Illegal characters that occur in the source data are converted to escaped entities, such as > for >, on the output side.
<i>Always</i>	Specifies that whatever form the source data takes, it will get wrapped in a CDATA section on output.

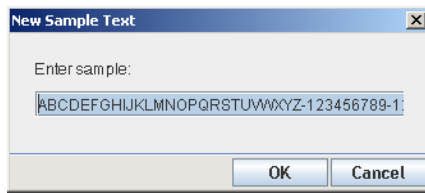
Transforming Elements with the Content Editor

You use the Content Editor to change the format and content of the input element to match that required by the output element. Using the Content Editor, you can slice the input data into small parts, move the parts to different locations relative to one another, add new parts, omit some parts, and apply functions to individual parts.

- 1 In the Action model, select two elements from different parts (for example, from the Input and Output parts) to map.
- 2 Select *New Action > Map*.
- 3 In the Map action dialog box (see [“Adding a Map Action” on page 242](#)), select the *Content Editor* check box, then select the *Content Editor* button.



- 4 If desired, click *New Sample* and type a sample string.



- 5 Click *OK* to return to the *Content Editor* dialog box.
- 6 In the *Sample* section, move the slider that is above the sample to the position where you want the first cut to take place, then move the slider that is below the sample to the position where you want the end cut to take place.

The sliders determine how to take a substring from the input data.

- 7 Click *Apply*.

The substring is copied to the *Result* field as a separate object.

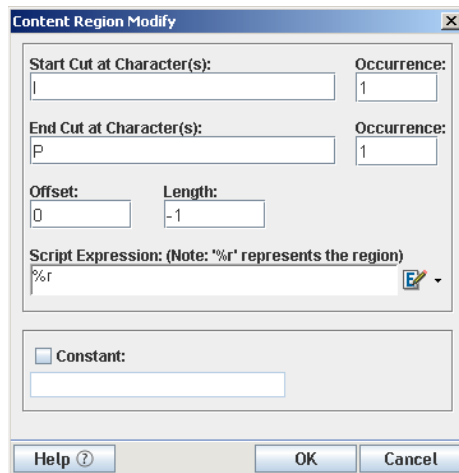
- 8 Repeat **Step 6** and **Step 7** for each part of the sample in the order that you want.

Using this method, you can build a new string out of substrings of the original input.

- 9 To change the format of an object in the *Result* field:

9a Select an object.

9b Click *Modify*.



The *Start Cut at Characters* field displays the character in the string where the first cut will take place. The first *Occurrence* field displays when the cut will take place. In the previous illustration, the first cut will take place at the first occurrence of the letter l. The *End Cut at Characters* field displays that character in the string where the last cut will take place. The second *Occurrence* field displays when the cut will take place. The *Offset* field displays the number of characters from the beginning of the original string where the object will start. The *Length* field displays the length of the object.

- 9c If desired, you can write an ECMAScript expression in the *Script Expression* field to modify the content region.

The *%r* shown in the *Script Expression* field is a local variable representing the content region to which you would like to apply a function. For example, to apply the

.toUpperCase() function to the content region, you would write the Script Expression: `var test='%r'; test.toUpperCase()`.

- 9d** To assign a constant to an object, select *Constant*, then type a constant string.
- 9e** When you are finished mapping string formats with the Content Editor, click *OK* to save the changes and close the Content Editor.
- 10** Click *OK* to return to the action model.

This section provides details on using the ECMA expression builder. Topics include:

- ♦ [Section 10.1, “About the ECMA Expression Builder,” on page 249](#)
- ♦ [Section 10.2, “ECMAScript Examples,” on page 259](#)
- ♦ [Section 10.3, “ECMAScript API,” on page 262](#)

10.1 About the ECMA Expression Builder

Designer incorporates an ECMAScript interpreter and expression editor, which allows you create script expressions that refer to and modify workflow data. For example, you can use scripting to:

- ♦ Create new data items needed in a workflow under the flowdata element.
- ♦ Perform basic string, date, math, relational, concatenation, and logical operations on data.
- ♦ Call standard or custom Java classes for more sophisticated data operations.
- ♦ Use expressions for runtime control to
 - ♦ Modify or override form field labels.
 - ♦ Initialize form field data.
 - ♦ Customize e-mail addresses and content.
 - ♦ Set entitlement grant/revoke rights and parameters.
 - ♦ Evaluate any past activity data to conditionally follow a work flow path using the Condition activity.
 - ♦ Write different log messages that are conditionally triggered using a single Log activity.

This section describes some of the techniques and capabilities applicable to the use of scripting.

NOTE: To define expressions for a workflow, you need to understand how workflow activities are configured. In addition, you need to understand the various types of data that are available within a workflow. For details on configuring workflow activities, see [Chapter 8, “Workflow Activity Reference,” on page 179](#). For descriptions of the system variables available in a workflow, see [Section 4.3.3, “Understanding Workflow Data,” on page 86](#).

10.1.1 About ECMAScript

ECMAScript is an object-oriented scripting language for manipulating objects in a host environment (in this case, Designer). ECMAScript (ECMA-262 and ISO/IEC 16262) is the standards-based scripting language underpinning both JavaScript* (Netscape*) and JScript* (Microsoft*). It is designed to complement and extend existing functionality in a host environment such as Designer’s graphical user interface. As a host environment, Designer provides ECMAScript access to various objects for processing. ECMAScript in turn provides a Java-like language that can operate on those objects.

The extensibility of ECMAScript, and its powerful string-handling tools (including regular expressions), make it an ideal language to extend the functionality of Designer.

NOTE: You can find detailed information about ECMAScript at the [European Computer Manufacturers Association \(ECMA\) Web site \(http://www.ecma-international.org\)](http://www.ecma-international.org).

10.1.2 ECMAScript Capabilities

In addition to enabling you to incorporate finely-tuned custom logic into your workflow, scripting gives you a great deal of flexibility in manipulating data because of the various DOM- and XPath-related objects and methods available in the ECMAScript extensions incorporated into the Expression Builder.

The usefulness of ECMAScript is especially apparent when dealing with in-memory DOMs. The ECMA expression builder constructs XML documents as in-memory objects according to the W3C DOM Level 2 specification. The DOM-2 specification, in turn, defines an ECMAScript binding (see the W3C Recommendation [ECMAScript Language Binding \(http://www.w3.org/TR/DOM-Level-2-Core/ecma-script-binding.html\)](http://www.w3.org/TR/DOM-Level-2-Core/ecma-script-binding.html), with numerous methods and properties that provide ready access to DOM-tree contents. The flowdata DOM is recognized by the ECMA expression builder, and any of the W3C-defined ECMAScript extensions that apply to DOMs can be used.

ECMAScript also provides bridges to other expression languages such as XPath. This allows you to use XPath syntax on a DOM to address various elements within its document structure.

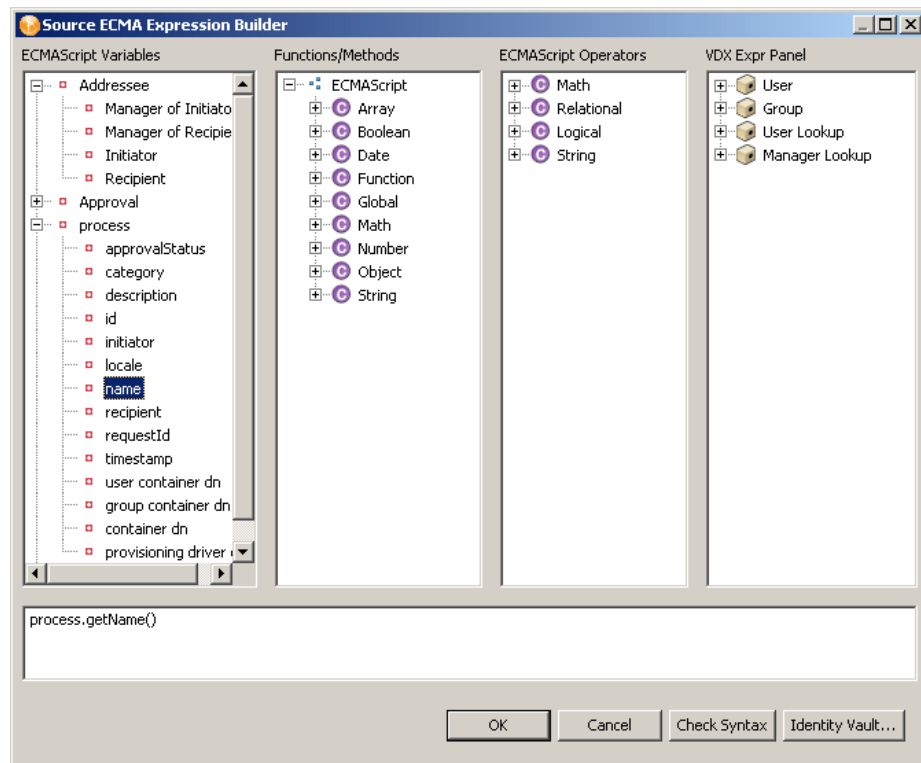
10.1.3 Using the ECMA Expression Builder

Designer provides access to ECMAScript in various places in the User Application design tools. The most common form of access is through the Expression Builder, which can be displayed whenever you see this button:



The button can be found in Designer displays, such as the Properties for a Condition activity or the Data Item Mapping view for an Entitlement Provisioning activity. Click the button to display the ECMA expression builder.

Figure 10-1 *ECMA Expression Builder*



The ECMA expression builder provides pick lists of available objects, methods, and properties in the top panes (all of which are resizable), with rollover tool tips to help you build ECMAScript statements. Double-clicking any item in any pick list causes a corresponding ECMAScript statement to appear in the edit pane in the lower portion of the window. In the figure, the *process* pick list has been selected in the *ECMAScript Variables* pane, and the *name* variable has been double-clicked. The ECMAScript expression that can access the contents of this workflow variable is inserted automatically in the edit pane.

This section includes the following topics:

- ◆ [“Checking Syntax” on page 252](#)
- ◆ [“Selecting a DN” on page 252](#)
- ◆ [“ECMAScript Variables” on page 252](#)
- ◆ [“Functions/Methods” on page 253](#)
- ◆ [“ECMAScript Operators” on page 253](#)
- ◆ [“VDX Expr” on page 255](#)
- ◆ [“Using Special Characters” on page 255](#)

Checking Syntax

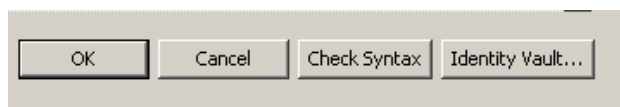
The ECMA expression builder includes a *Check Syntax* button. Clicking the button causes the ECMAScript interpreter to check the syntax of the expression. If there are problems involving ECMAScript syntax, an error message is displayed. You can then edit the expression and validate again as needed. Validation is optional.

NOTE: The syntax checking process does not execute your expression. It just checks syntax. Because ECMAScript is an interpreted language, syntax checking doesn't check any runtime-dependent expressions, other than to see if they conform to valid ECMAScript syntax.

Selecting a DN

The ECMA expression builder also includes an *Identity Vault* button that is displayed when you are working with activities that might require selecting a DN from the Identity Vault (for example, Start, Approval, and Entitlement activities).

Figure 10-2 *Identity Vault Button*



The *Identity Vault* button displays a dialog box that you use to navigate the Identity Vault to select a DN. The Identity Vault button is grayed (to indicate that it is unavailable) if you are not connected to the Identity Vault.

ECMAScript Variables

This pane displays the names of variables that are relevant in the current context. For example, if you are working in the provisioning request definition editor, you see system variables for the current workflow process, system variables for the current activity, and flowdata variables created in the current workflow. Double-click the name of a variable to insert that variable into your script. For descriptions of the system variables available in a workflow, see [Section 4.3.3, “Understanding Workflow Data,” on page 86](#).

The ECMA expression builder provides two methods for reading flowdata variables.

Table 10-1 *Methods for Reading Flowdata Variables*

Method	Description
<code>flowdata.get(variable-name)</code>	Returns a string as the node value for a variable (representing an XPath expression) in the workflow document.
<code>flowdata.getObject(variable-name)</code>	Returns an object as a node value for a variable (representing an XPath expression) in the workflow document. Use this method to retrieve the values of multivalued controls.

Functions/Methods

For a description of the functions and methods available in the ECMA expression builder, see [Section 10.3, “ECMAScript API,” on page 262](#).

ECMAScript Operators

The following tables provide descriptions of the operators supported by the ECMA expression builder.

Table 10-2 *Math*

Operator	Description
+ Add	Returns the sum of two numerical values (either literals or variables).
- Subtract	Subtracts one number from another .
* Multiply	Returns the product of two numerical values (either literals or variables).
/ Divide	Divides one number by another.

Table 10-3 *Assignment*

Operator	Description
= Assignment	Assigns the value of the right operand to the left operand.
+= Add to this	Adds the left and right operands and assigns the result to the left operand. For example, <code>a += b</code> is the same as <code>a = a + b</code> .
-= Subtract from this	Subtracts the right operand from the left operand and assigns the result to the left operand. For example, <code>a -= b</code> is the same as <code>a = a - b</code> .
*= Multiply to this	Multiplies the two operands and assigns the result to the left operand. For example, <code>a *= b</code> is the same as <code>a = a * b</code> .
/= Divide this to	Divides the left operand by the right operand and assigns the result to the left operand. For example, <code>a /= b</code> is the same as <code>a = a / b</code> .
%= Modulus	Divides the left operand by the right operand and assigns the remainder to the left operand. For example, <code>a %= b</code> is the same as <code>a = a % b</code> .
&= Apply bitwise AND to this	Performs bitwise AND on operands and assigns the result to the left operand. For example, <code>a &= b</code> is the same as <code>a = a & b</code> .
= Apply bitwise OR to this	Performs bitwise OR on operands and assigns the result to the left operand. For example, <code>a = b</code> is the same as <code>a = a b</code> .
<<= Apply bitwise left shift to this	Performs bitwise left shift on operands and assigns the result to the left operand. For example, <code>a <<= b</code> is the same as <code>a = a << b</code> .

Operator	Description
>>= Apply bitwise signed right shift to this	Performs bitwise right shift on operands and assigns the result to the left operand. For example, <code>a >>= b</code> is the same as <code>a = a >> b</code> .
>>>= Apply bitwise unsigned right shift to this	Performs bitwise unsigned right shift on operands and assigns the result to the left operand. For example, <code>a >>>= b</code> is the same as <code>a = a >>> b</code> .

Table 10-4 *Other*

Operator	Description
% Modulus	Divides the left operand by the right operand and returns the integer remainder.
++ Autoincrement	Increments the operand by one (can be used before or after the operand).
-- Autodecrement	Decrements the operand by one (can be used before or after the operand).
~ Bitwise NOT	Inverts the bits of its operand.
& Bitwise AND	Returns a 1 in each bit position for which the corresponding bits of both operands are ones.
Bitwise OR	Returns a 1 in each bit position for which the corresponding bits of either or both operands are ones.
^ Bitwise XOR	Returns a 1 in each bit position for which the corresponding bits of either but not both operands are ones.
<< Bitwise left shift	Shifts the digits of the binary representation of the first operand to the left by the number of places specified by the second operand. The spaces created to the right are filled in by zeros, and any digits shifted to the left are discarded.
>> Signed bitwise right shift	Shifts the digits of the binary representation of the first operand to the right by the number of places specified by the second operand, discarding any digits shifted to the right. The copies of the leftmost bit are added on from the left, preserving the sign of the number.
>>> Unsigned bitwise right shift	Shifts the binary representation of the first operand to the right by the number of places specified by the second operand. Bits shifted to the right are discarded and zeroes are added to the left.

Table 10-5 *Relational*

Operator	Description
== Equal	Assigns the value of the right operand to the left operand.
!= Not Equal	Returns a Boolean true if the operands are not equal.

Operator	Description
< Less than	Returns true if the left operand is less than the right operand.
> Greater than	Returns true if the left operand is greater than the right operand.
<= Less than or equal	Returns true if the left operand is less than or equal to the right operand.
>= Greater than or equal	Returns true if the left operand is greater than or equal to the right operand.

Table 10-6 *Logical*

Operator	Description
&& AND	Returns a Boolean true if both operands are true; otherwise, returns false.
OR	Returns true if either operand is true. Returns false when both operands are false.
! NOT	Returns false if its single operand can be converted to true (or if it is a non-Boolean value). Returns true if its operand can be converted to false.

Table 10-7 *String*

Operator	Description
+ Concatenate	Concatenates two string operands, returning a string that is the union of the two operand strings.

VDX Expr

This pane allows you to insert Entity definitions (see [Section 3.2, “Working with Entities and Attributes,” on page 41](#)) that are defined in the Identity Vault into your scripts. Both system and user-defined entities are available. The format of an expression to retrieve data from the Identity Vault is

```
IDVault.get(dn, object-type, attribute)
```

For example if you want the recipient's manager for a data item, you would open the User node in the VDX Expr Panel and double-click the Manager item, which inserts `IDVault.get({ enter dn expression here }, 'user', 'manager')`. This expression evaluates to the string for the manager's DN (LDAP distinguished name).

Using Special Characters

You can use special characters in literal strings in the ECMA expression builder by using escape sequences. Escape sequences begin with a backslash character (`\`). The following table contains some commonly-used escape sequences:

Table 10-8 *Escape Sequences*

Escape Sequence	Character
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\"	Double quote
\\	Backslash (\)
\'	Apostrophe

You also can specify any Unicode character by using '\u ' followed immediately by four hexadecimal digits. Here are some examples:

Table 10-9 *Escape Sequence Examples for Unicode Characters*

Escape Sequence	Character
\u00A3	Pound symbol (£)
\u20AC	Euro symbol (€)

10.1.4 About Java Integration

Java is integrated into the workflow process through the ECMA expression builder, which provides a bridge to external Java objects. To access a Java class through the ECMA expression builder, the class must be in the classpath of the workflow engine. To accomplish this, you must add the Java class to the `WEB-INF\lib` directory in the User Application WAR file (`IDM.war`).

NOTE: Unlike ECMAScript that is available in other parts of the provisioning request definition editor, form action scripts are executed on the browser, not the server. All directory access from within a form action script is handled by AJAX calls from the browser to the server. See [Section 10.3.1, “Form Action Script Methods,” on page 262](#).

Adding the Java Class to the User Application WAR

- 1 Use a WAR file utility to open the `IDM.war` file. The `IDM.war` file is located in the application server `\server\IDM\deploy` directory.
- 2 Copy the Java class into the `WEB-INF\lib` directory.

Accessing Java from ECMAScript

To access a Java class, create a function inline in the ECMA expression builder. Instantiate the function, then within the function, using ECMAScript syntax, call your Java methods. The following example creates a vector:

```
function list() { v=new java.util.Vector(); v.add('{Enter Item 1}');  
v.add('{Enter Item 2}'); return v; }; list();
```

To access a custom Java class, you must preface the class name with “Packages”. For example:

```
v = new Packages.com.novell.myClass("value");
```

The ECMA expression builder is built on Mozilla* Rhino. Rhino is an open-source implementation of JavaScript written entirely in Java. For more information about accessing Java from ECMAScript, see [Scripting Java \(http://www.mozilla.org/rhino/ScriptingJava.html\)](http://www.mozilla.org/rhino/ScriptingJava.html).

10.1.5 About XPath Integration

This section includes the following topics:

- ♦ “XPath in Workflows” on page 257
- ♦ “XPath in Integration Activities” on page 257

XPath in Workflows

A provisioning request definition workflow supports a special object called *flowdata* (see [Section 4.3.3, “Understanding Workflow Data,” on page 86](#)). The flowdata object is a DOM (an XML document constructed as an object in memory). You can use XPath syntax to navigate the structure of the flowdata DOM, and add, modify or delete elements and contents.

To add an object to flowdata:

Syntax	Examples
<pre>flowdata.{arguments}</pre> <div><div>ECMAScript</div><div>XPath</div></div>	<pre>flowdata.parent/child[1] flowdata.reason</pre>

To get an object from flowdata:

Syntax	Examples
<pre>flowdata.get{arguments}</pre> <div><div>ECMAScript</div><div>XPath</div></div>	<pre>flowdata.getObject('parent/ child[1]') flowdata.get('reason')</pre>

For information about the flowdata.get() and flowdata.getObject() methods, see [Table 10-1 on page 252](#).

XPath in Integration Activities

When you are working with an Integration activity, the ECMAScript interpreter recognizes a custom method called XPath(). This method allows expressions such as:

```
Input.XPath ("GetBNQuoteSoapIn/GetBNQuote/sISBN")
```

Using the ECMA expression builder, this type of expression is built for you automatically when you select nodes in ECMA expression builder pick lists.

The Integration activity uses the XPath addressing syntax adopted by W3C. The XPath syntax is similar to URI address syntax but includes many subtle and powerful features for addressing and manipulating XML. Some of the more common syntax rules are listed in the following table.

Table 10-10 *XPath Syntax*

XPath Syntax	Description
/	The single forward slash represents an absolute path to an element. For example, /ABC selects the root element ABC.
//	Double slashes represent all elements in a path. //ABC selects all occurrences of ABC. For example, //ABC//DEF selects all DEF elements that are children of ABC.
*	The asterisk selects all elements located by the preceding path. For example, *ABC/DEF selects all elements enclosed by elements ABC/DEF. //* selects all elements.
[]	Square brackets specifies a particular element. For example /ABC[3] selects the third element in ABC. This can also be used as a filter (similar to a Where clause in SQL). //ABC["Table"] selects all elements that have the content "Table."
@	The At sign selects elements with a specified attribute. For example, /ABC@name selects all elements in ABC that have an attribute called name.
	The vertical bar allows you to specify multiple paths. For example, //ACB //DEF selects all elements in ACB and in DEF.
\$	The dollar sign allows you to reference other documents besides the current one: INVOICEBATCH/INVOICE[SELLER/NAME=\$PROJECT/USERCONFIG/COMPANYNAME]
function()	XPath has numerous functions that you can add to your XPath addresses. For example, //*[count(*)=2] selects all elements that have two children.
math operator()	XPath has numerous math operators that you can add to your XPath addresses. For example, /ABC[position() mod 2 = 0] selects all even elements in ABC.

You can find the complete list of operators in the W3 Recommendation [XML Path Language \(XPath\)](http://www.w3.org/TR/xpath.html) (<http://www.w3.org/TR/xpath.html>).

10.1.6 Performance Considerations

ECMAScript is an interpreted language, which means that every line of script in an expression must be parsed and translated to the Java equivalent before it can be executed. This adds considerable overhead to the code and results in overall slower execution of scripts than pure Java. Before using ECMAScript, you should think about the possible performance ramifications.

The following guidelines will help you to achieve optimal performance in your components and services:

- ♦ Consider whether a task can be accomplished using a custom Java class (which you can call from ECMAScript).
- ♦ When you need the fine control offered by scripting, use ECMAScript.

Bear in mind that the key to good performance is always a good implementation (for example, choosing the correct algorithm and attention to reuse of variables). Good code written in a slow language often outperforms bad code written in a fast language. Writing something in Java does not guarantee that it will be faster than the equivalent logic written in ECMAScript because Java has its own overhead constraints, for example, constructor call-chains (when you call a constructor for a Java object that inherits from other objects, the constructors for all ancestral objects are also called).

ECMAScript's core objects (String, Array, Date, etc.) have many built-in convenience methods for data manipulation, formatting, parsing, sorting, and conversion of strings and arrays. These methods are implemented in highly optimized Java code inside the interpreter. It is to your advantage to use these methods whenever possible, rather than create customized data-parsing or formatting functions. For example, suppose you want to break a long string into substrings, based on the occurrence of a delimiter. You could create a loop that uses the String methods `indexOf()` and `substring()` to parse out the substrings and assign them to slots in an array. But this would be a very inefficient technique when you could simply do the following:

```
var myArrayOfSubstrings = bigString.split( delimiter );
```

The ECMAScript String method `split()` breaks a string into an array of substrings based on whatever delimiter value you supply. It executes in native Java and requires the interpreter to interpret only one line of script. Trying to do the same thing with a loop that iteratively calls `indexOf()` and `substring()` would involve a great deal of needless interpreter and function-call overhead, with the accompanying performance hit.

Skillful use of built-in ECMAScript methods pays worthwhile performance dividends. If you use scripts extensively, take time to learn about the fine points of the ECMAScript language because this can help you eliminate performance bottlenecks.

10.2 ECMAScript Examples

This section provides examples of common operations that you can perform using ECMAScript.

10.2.1 General Examples

This section presents examples that illustrate basic scripting techniques.

Using a Function

To create a function in the ECMA expression builder, create the function inline:

```
function abc() { var v1 = "" ; for ( i = 0; i < 9 ; i++) v1 += "$";  
return v1; } ; abc();
```

10.2.2 Flowdata Examples

This section presents scripting examples that show the use of the flowdata object.

Getting the Value of a Flowdata Variable

In the previous example, you entered information about an approval status into the flowdata by creating an XML element named `start_reason` with a child element named `approval_reason` and an attribute named `ApprovalStatus`. Use the following expression, in a pre-activity map, to retrieve the value of the `ApprovalStatus` attribute:

```
flowdata.get('start_reason/approval_reason/@ApprovalStatus')
```

You can enter this expression by expanding the *flowdata* nodes in the ECMAScript Variables pane of the ECMA expression builder and double-clicking the `ApprovalStatus` attribute.

Figure 10-3 *Selecting an Attribute*



Creating an XML Element with Child Element and Adding it to the Flowdata

In the previous example, you retrieved user input to the form field `ApprovalStatus`. Now we want to add this information to the flowdata so that it can be used by a downstream activity. Use the following expression in a post-activity map:

```
flowdata.start_reason/approval_reason/@ApprovalStatus
```

10.2.3 Form Control Examples

This section presents several examples of scripting with form controls.

Retrieving the Value of a Form Field

Suppose you have a form field named `ApprovalStatus`. To get the value of this field, use the following expression in a pre-activity map:

```
process.get('ApprovalStatus')
```

You can enter this expression by opening the Process node in the ECMAScript Variables pane of the ECMA expression builder and double-clicking `ApprovalStatus`.

Getting an Individual Value from a Multivalued Control

To get an individual value from a multivalued control (for example, a check box named *colors*), you first need to get the control into the flowdata. In the post-activity mapping for an upstream activity, use the following:

```
flowdata.colors
```

To get a value from `colors` (for example, the first value), use the following expression on a downstream activity:

```
flowdata.getObject('colors[1]')
```

Populating a List or Checkbox Item

To populate list controls (for example, PickList or MVEditor) or the MVCheckbox control using script, use an expression like this in the pre-activity mapping:

```
function list() {var l=new
java.util.Vector();l.add('Blue');l.add('Red'); l.add('Green'); return
l;} list();
```

Comparing DNs

To compare DNs to find out if they are equal, use an expression like this:

```
if ( IDVault.DNcompare(flowdata.get('Activity3/CardRequest/
Candidate'),recipient )) true; else false ;
```

This comparison is case-insensitive. For example, the following DNs, when compared with DNCompare, would return true:

```
CN=jdoe,ou=users,ou=idmsample,o=acme
cn=JDOE,ou=users,ou=idmsample,o=acme
```

10.2.4 Error Handling Examples

This section presents scripting examples that show how to deal with errors during runtime execution.

Handling Errors

The approach to handling errors differs between pre-activity and post-activity maps. For post-activity maps, you can use an error flow path from an Approval or Condition activity to catch errors that occur during post-activity mapping. This approach doesn't work for pre-activity maps because any errors that occur in the process of getting data happen before the form is displayed to the user. When this occurs, an error message similar to the following appears in place of form controls in the bottom portion of forms displayed to the user:

```
XXXX FAILED to generate form due to: No data items are available!
```

In this scenario, you can use a try-catch statement in a source expression for a field in a pre-activity map:

```
function getTheData()
{
    var theData;
    try {
        theData = IDVault.get( 'cn=jsmith,ou=users,ou=idmsample1,o=acme'
, 'user', 'FirstName') + ' ' + IDVault.get (
'cn=jsmith,ou=users,ou=idmsample1,o=acme' , 'user', 'LastName');
    }
    catch (error) { theData = 'Error retrieving data.'; }
    return theData;
};
getTheData();
```

10.3 ECMAScript API

This section includes the following topics:

- ♦ [Section 10.3.1, “Form Action Script Methods,” on page 262](#)
- ♦ [Section 10.3.2, “DOM Methods,” on page 269](#)
- ♦ [Section 10.3.3, “ECMAScript Core,” on page 293](#)
- ♦ [Section 10.3.4, “Global Functions,” on page 312](#)
- ♦ [Section 10.3.5, “IDVault Functions,” on page 312](#)

10.3.1 Form Action Script Methods

Unlike the ECMAScript that runs in other components of the workflow, form script executes on the Web browser, not the server. All directory access from within form script is handled by AJAX* calls from the browser to the server.

This section lists all form action methods and properties supported by the ECMA expression builder. This section includes the following topics:

- ♦ [“Form” on page 262](#)
- ♦ [“Field” on page 265](#)
- ♦ [“Event” on page 267](#)
- ♦ [“Lists” on page 267](#)
- ♦ [“Queries” on page 268](#)
- ♦ [“Container” on page 268](#)

Form

Lets you work with Form methods. This section includes the following methods:

- ♦ [“alert\(string\)” on page 262](#)
- ♦ [“showMsg\(string\)” on page 263](#)
- ♦ [“showWarning\(string\)” on page 263](#)
- ♦ [“showError\(string\)” on page 263](#)
- ♦ [“showFatal\(string\)” on page 264](#)
- ♦ [“enable\(fieldname\)” on page 264](#)
- ♦ [“disable\(fieldname\)” on page 264](#)
- ♦ [“getValue\(fieldname\)” on page 264](#)
- ♦ [“getValues\(fieldname\)” on page 265](#)
- ♦ [“setValues\(fieldname\)” on page 265](#)

alert(string)

```
form.alert("msg")
```

Displays a message in an alert box.

showMsg(string)

```
form.showMsg(msg, param, bId)
```

Adds a message to the status portion of the form. The `msg` string parameter can either contain the text of the message itself or it can contain a key pointing to an entry in the resource bundle `bId`. This method always tries to find an entry with the key `msg` in the resource bundle with the id `bId`. The `param` parameter can be used to pass in replacements for stakeholders (`{0}`, `{1}`, etc) in `msg`.

NOTE: If you want to add debugging messages to your script, it is better practice to use `form.showDebugMsg()` .

Example:

```
form.showMsg("my message" {0},{1}", ["value0","value1"]);
```

showWarning(string)

```
form.showWarning(msg, param, bId)
```

Adds a warning to the status portion of the form.

The `msg` string parameter can either contain the text of the warning itself or it can contain a key pointing to an entry in the resource bundle `bId`. This method always tries to find an entry with the key `msg` in the resource bundle with the id `bId`. The `param` parameter can be used to pass in replacements for stakeholders (`{0}`, `{1}`, etc) in `msg`.

NOTE: If you want to add debugging messages to your script, it is better practice to use `form.showDebugMsg()` .

Example:

```
form.showWarning("my warning" {0},{1}", ["value0","value1"]);
```

showError(string)

```
showError(msg, param, bId);
```

Adds an error message to the status portion of the form.

The `msg` string parameter can either contain the text of the error itself or it can contain a key pointing to an entry in the resource bundle `bId`. This method always tries to find an entry with the key `msg` in the resource bundle with the id `bId`. The `param` parameter can be used to pass in replacements for stakeholders (`{0}`, `{1}`, etc) in `msg`.

NOTE: If you want to add debugging messages to your script, it is better practice to use `form.showDebugMsg()` .

Both normal and fatal errors block form submission. The distinction between a normal error and a fatal error is that normal errors get reset just before form validation occurs (due to a form submission). Fatal errors are remembered and therefore block the form submission unless you restart. A normal error only blocks submission if it is generated during the validation phase. If you add normal errors during onload or custom events, they will be lost when the form is submitted.

NOTE: If you want to add debugging messages to your script, it is better practice to use `form.showDebugMsg()` .

Example:

```
form.showError("my error" {0},{1}", ["value0","value1"]);
```

showFatal(string)

```
form.showFatal("my fatal" {0},{1}", ["value0","value1"]);
```

Adds an fatal error message to the status portion of the form.

The `msg` string parameter can either contain the text of the fatal error itself or it can contain a key pointing to an entry in the resource bundle `bId`. This method always tries to find an entry with the key `msg` in the resource bundle with the id `bId`. The `param` parameter can be used to pass in replacements for stakeholders (`{0}`, `{1}`, etc) in `msg`.

Both normal and fatal errors block form submission. The distinction between a normal error and a fatal error is that normal errors get reset just before form validation occurs (due to a form submission). Fatal errors are remembered and therefore block the form submission unless you restart. A normal error only blocks submission if it is generated during the validation phase. If you add normal errors during onload or custom events, they will be lost when the form is submitted.

NOTE: If you want to add debugging messages to your script, it is better practice to use `form.showDebugMsg()` .

Example:

```
form.showFatal("my fatal" {0},{1}", ["value0","value1"]);
```

enable(fieldname)

```
form.enable("fieldname")
```

Enables a field on a form.

disable(fieldname)

```
form.disable("fieldname")
```

Disables a field on a form.

NOTE: A disabled field still sends data back to the workflow engine. The content of a disabled field is validated when submitting the form or when calling the `field.validate()` method .

getValue(fieldname)

```
form.getValue("fieldname")
```

Returns the first value for the field. The type returned is always string, independent of the data type of the field. If the field does not have a value, the method returns an empty string if text can be entered into the field (like Text, TextArea, DatePicker, DNLookup) or it returns “undefined” if the control is choice-based (for example, StaticList, radio buttons, check boxes). For DN type controls, this method always returns the DN and never the display expression.

getValues(fieldname)

```
form.getValues("fieldname")
```

Returns a string array containing the values. If no values are found, the array is empty (size = 0). For DN type controls, this method always returns the DN and never the display expression.

setValues(fieldname)

```
form.setValues("fieldname", data-values, display values,  
KeepOldValues)
```

Sets a value. Supports multiple values. This method allows changing the available entries for list-based controls (for example, StaticList, MVCheckbox, PickList). By default, existing values are deleted unless the `KeepOldValues` parameter equals true. For non-list-based controls, the `display values` parameter is ignored.

If you want to set or change the initial value of a field, you should do so in an “onload” event.

NOTE: This method triggers the onchange event for the field.

Examples:

```
field.setValues("cn=jdoe,ou=users,ou=mysample,o=novell"); // for a  
DNLookup  
field.setValues(["jdoe@novell.com", "test@novell.com"]) // for an  
MVEditor  
field.setValues(["W", "B"], ["White", "Black"], true); // for a StaticList
```

Field

Lets you work with Field methods. This section includes the following methods:

- ♦ “getLabel()” on page 265
- ♦ “fireEvent()” on page 265
- ♦ “getValue()” on page 266
- ♦ “getValues()” on page 266
- ♦ “setValues(fieldname)” on page 266
- ♦ “enable()” on page 266
- ♦ “disable()” on page 266

getLabel()

```
field.getLabel()
```

Gets the label associated with the field. If no label is found, this method returns the name of the field.

fireEvent()

```
field.fireEvent("eventname")
```

Fires a custom event. Passes the name of the custom event that is fired. To get the values of the event that is fired, use `form.getValues(event.getOrigin())`.

getValue()

```
field.getValue()
```

Returns the first value for the field. The type returned is always a string, independent of the data type of the field. If the field does not have a value, the method returns an empty string if text can be entered into the field (like Text, TextArea, DatePicker, DNLookup) or it returns “undefined” if the control is choice-based (for example, StaticList, radio buttons, check boxes). For DN type controls, this method always returns the DN and never the display expression.

getValues()

```
form.getValues()
```

Returns a string array containing the requested values. If no values are found, the array is empty (size = 0). For DN type controls this method always returns the DN and never the display expression.

setValues(fieldname)

```
field.setValues(data-values, display-values, KeepOldValues)
```

Sets a value. Supports multiple values. This method allows changing the available entries for list-based controls (for example, StaticList, MVCheckbox, PickList). By default, existing values are deleted unless the `KeepOldValues` parameter equals true. For non-list-based controls, the `display values` parameter is ignored.

If you want to set or change the initial value of a field, you should do so in “onload” event.

NOTE: This method triggers the onchange event for the field.

Examples:

```
field.setValues("cn=jdoe,ou=users,ou=mysample,o=novell"); // for a
DNLookup
field.setValues(["jdoe@novell.com", "test@novell.com"]) // for an
MVEditor
field.setValues(["W", "B"], ["White", "Black"], true); // for a StaticList
```

enable()

```
field.enable()
```

Enable the field.

disable()

```
field.disable()
```

Disable the field.

NOTE: A disabled field still sends data back to the workflow engine. The content of a disabled field is validated when submitting the form or when calling the field.

Event

Lets you work with events. This section includes the following methods:

- ♦ “[getOrigin\(\)](#)” on page 267
- ♦ “[getValue\(\)](#)” on page 267
- ♦ “[getValues\(\)](#)” on page 267

getOrigin()

```
event.getOrigin()
```

Returns the name of the field from which the event was triggered.

getValue()

```
event.getValue()
```

Returns a string that contains the first value in the event.

You should not use the value that is returned by this method, since it is possible that a user might have modified the data in the field since the event was triggered. Instead, you should use the value returned by the `form.getValue` method. For example, `form.getValue(event.GetOrigin())`. This ensures that you get the current value of the field. If you select `event.getValue()` from the pick list in the ECMA expression builder, `form.getValue(event.GetOrigin())` is inserted.

getValues()

```
event.getValues()
```

Returns an array of strings that contains all values in the event.

You should not use the value that is returned by this method, since it is possible that a user might have modified the data in the field since the event was triggered. Instead, you should use the value returned by the `form.getValues` method. For example, `form.getValues(event.GetOrigin())`. This ensures that you get the current value of the field. If you select `event.getValues()` from the pick list in the ECMA expression builder, `form.getValues(event.GetOrigin())` is inserted.

Lists

Lets you work with lists.

globalList(fieldname, key, locale)

```
IDVault.globalList("fieldname", "key", "locale")
```

Retrieves a global list from the directory abstraction layer, identified by the key of the global list. If the field name is specified, the result of the query is used to refresh the content of the field. To retrieve a list without storing the result in a field, use a null value for the fieldname parameter.

The locale is optional. If locale is not specified, the locale in the HTTP request is used.

Example:

```
IDVault.globalList("dallist", "departments", "en");
```

Queries

Lets you work with queries.

globalQuery(fieldname, key, param)

`globalQuery(fieldname, key, param)`

Executes the predefined directory abstraction layer query key (see “[Queries General Properties](#)” on [page 73](#)). If the field name is specified, the result of the query is used to refresh the content of the field. To retrieve a list without storing the result in a field, use a null value for the fieldname parameter.

The param parameter is used as input to the query. The parameter has the form {parname1:value, parname2:value}, in which the value can be an individual value or an array. The first column of the result list (always a DN) is used for the data value, the second column is used for the display label.

Example:

```
IDVault.globalQuery("canchangepwd", "getsites"); // query without a
parameter
IDVault.globalQuery("building", "getbuildings",
{site:form.getValue("site")}); // query with one parameter
IDVault.globalQuery("room", "getrooms", {site:form.getValue("site"),
building:form.getValue("building")}); // query with two parameters
```

Container

Lets you work with containers.

containers(fieldname, rootdn, Search scope, Show DN)

`IDVault.containers("test", rootdn, SearchScope, ShowDN)`

Gets a list of containers, with the scope equal to “subtree” or the same level. The method returns an array with 2 entries, the first an array with the resulting DNs; the second entry an array with the display labels.

Table 10-11 Container Parameters

Parameter	Description
fieldname	If the field name is specified, the result of the query is used to refresh the content of the field. To retrieve a list without storing the result in a field, use a null value for the fieldname parameter.
rootdn	If the rootdn parameter is empty, the root container for the default entity is used.
scope	If the scope parameter is empty, one-level is used. Valid choices for scope are “o” (nelevel) and “s” (ubtree).
showdn	If the parameter showDN is true, the full DN is used for the display label. Otherwise the naming part (for example, ou, dc) is displayed.

Example:

```
IDVault.containers("assetProp2", null, "o", true); // set the entries
in a StaticList to all containers directly under the root DN of the
default entity
```

10.3.2 DOM Methods

This section lists all DOM-related methods and properties supported by the ECMA expression builder, including not only DOM-1 and DOM-2 extensions (defined by the relevant W3C standards), but also Designer's own ECMAScript extensions. Extension methods are specifically noted as such in the text. DOM methods are displayed in the ECMA expression builder when you are working with expressions in the Integration activity.

This section includes the following topics:

- ♦ [“Node” on page 269](#)
- ♦ [“Document” on page 273](#)
- ♦ [“Element” on page 278](#)
- ♦ [“Attribute” on page 284](#)
- ♦ [“CharacterData” on page 285](#)
- ♦ [“NodeList” on page 286](#)
- ♦ [“NamedNodeMap” on page 288](#)
- ♦ [“Text” on page 290](#)
- ♦ [“DocumentType” on page 290](#)
- ♦ [“DOMImplementation” on page 291](#)
- ♦ [“Notation” on page 292](#)
- ♦ [“Entity” on page 292](#)
- ♦ [“ProcessingInstruction” on page 293](#)

Node

Lets you work with nodes. This section includes the following topics:

- ♦ [“attributes” on page 270](#)
- ♦ [“childNodes” on page 270](#)
- ♦ [“firstChild” on page 270](#)
- ♦ [“lastChild” on page 270](#)
- ♦ [“nextSibling” on page 270](#)
- ♦ [“nodeName” on page 270](#)
- ♦ [“nodeType” on page 270](#)
- ♦ [“nodeValue” on page 271](#)
- ♦ [“ownerDocument” on page 271](#)
- ♦ [“parentNode” on page 271](#)
- ♦ [“previousSibling” on page 271](#)
- ♦ [“XML” on page 271](#)

- ♦ “appendChild(newChild)” on page 271
- ♦ “cloneNode(deep)” on page 271
- ♦ “createXPath(XPathType asPattern)” on page 271
- ♦ “hasChildNodes()” on page 271
- ♦ “insertBefore(newChild, refChild)” on page 272
- ♦ “removeChild(oldChild)” on page 272
- ♦ “replaceChild(newChild, oldChild)” on page 272
- ♦ “getXML()” on page 272
- ♦ “ownerDocument” on page 272
- ♦ “namespaceURI” on page 272
- ♦ “prefix” on page 272
- ♦ “localName” on page 272
- ♦ “normalize()” on page 273
- ♦ “hasAttributes()” on page 273
- ♦ “isSupported(feature, version)” on page 273

attributes

W3C DOM Level 1 Node property. This property returns a NamedNodeMap object of the attributes for the Node.

childNodes

W3C DOM Level 1 Node property. This property returns a NodeList object consisting of the immediate children of the Node.

firstChild

W3C DOM Level 1 Node property. This property returns the first child node of a Node object.

lastChild

W3C DOM Level 1 Node property. This property returns the last child node of a Node object.

nextSibling

W3C DOM Level 1 Node property. This property returns the next sibling node for a Node object.

nodeName

W3C DOM Level 1 Node property. This property returns the node name as a String object.

nodeType

W3C DOM Level 1 Node property. This property returns the node type as a short in which

- 1 = Element
- 2 = Attribute
- 3 = Text

4 = CDATASection
5 = EntityReference,
6 = Entity,
7 = ProcessingInstruction
8 = Comment
9 = Document
10 = DocumentType
11 = DocumentFragment
12 = Notation

nodeValue

W3C DOM Level 1 Node property. This property returns the node text data as a String.

ownerDocument

W3C DOM Level 1 Node property. This property returns a Document object.

parentNode

W3C DOM Level 1 Node property. This property returns the parent node object for a Node object.

previousSibling

W3C DOM Level 1 Node property. This property returns the previous sibling node for a Node object.

XML

Designer extension property. This property returns a string representing the DOM. Useful in Log actions for debugging components (for example, Input.XML).

appendChild(newChild)

`Node appendChild(newChild)`

W3C DOM Level 1 Node method. This method appends a node as the last child for a Node. The newChild parameter is of type Node.

cloneNode(deep)

`Node cloneNode(deep)`

W3C DOM Level 1 Node method. This method creates an unattached Node object. The deep parameter is of type boolean.

createXPath(XPathType asPattern)

`Object createXPath(XPathType asPattern)`

ECMAScript extension method. Creates the XPath pattern. The XPathType asPattern parameter only supports abbreviated XPath notation and explicit ordinals. XPath functions are not supported.

hasChildNodes()

`boolean hasChildNodes()`

W3C DOM Level 1 Node method. This method returns a boolean indicating whether the node has children.

insertBefore(newChild, refChild)

`Node insertBefore(newChild, refChild)`

W3C DOM Level 1 Node method. This method inserts a node object into the parent node before the `refChild` node. The `newChild` parameter is of type `Node`. The `refChild` parameter is of type `Node`.

removeChild(oldChild)

`Node removeChild(oldChild)`

W3C DOM Level 1 Node method. This method removes a node from a parent and returns an unattached node. The `oldChild` parameter is of type `Node`.

replaceChild(newChild, oldChild)

`Node replaceChild(newChild, oldChild)`

W3C DOM Level 1 Node method. This method replaces one node with another node. The `newChild` parameter is of type `Node`. The `oldChild` parameter is of type `Node`.

getXML()

`String getXML()`

ECMAScript extension method. This property returns a string representing the DOM. Useful in Log actions for debugging components. Example:

`Input.XPath("root/child").getXML()`

ownerDocument

W3C DOM Level 2 modified Node property. Returns the Document object associated with this node. This is also the Document object used to create new nodes. Example:

`someNodeObject.ownerDocument`

namespaceURI

W3C DOM Level 2 Node property. Returns the namespace URI of this node, or null if the namespace URI is not specified. Example:

`someNodeObject.namespaceURI`

prefix

W3C DOM Level 2 Node property. Returns the namespace prefix of this node, or null if the namespace prefix is not specified. Example:

`someNodeObject.prefix`

localName

W3C DOM Level 2 Node property. Returns the local part of the qualified name of this node. Example:

`someNodeObject.localName`

normalize()

```
void normalize()
```

W3C DOM Level 2 modified Node method. Puts all Text nodes in the full depth of the sub-tree underneath this Node, including attribute nodes, into a “normal” form in which only structure separates Text nodes, (for example, elements, comments, processing instructions, CDATA sections, and entity references). In other words, there are neither adjacent Text nodes nor empty Text nodes.

hasAttributes()

```
boolean hasAttributes()
```

W3C DOM Level 2 Node method. Returns true if the node has any attributes; otherwise, returns false. Example:

```
Temp.XPath("A/B/C").item(0).hasAttributes()
```

isSupported(feature, version)

```
boolean isSupported(feature, version)
```

W3C DOM Level 2 Node method. Returns true if the specified feature is supported on this node; otherwise, returns false.

Table 10-12 Parameters of IsSupported Method

Parameter	Features
feature	Core XML HTML Views Stylesheets CSS CSS2 Events UIEvents MouseEvents MutationEvents HTMLEvents Range Transversal
version	Specifies the version number of the feature to test. In Level 2, version 1, this is the string “2.0”. If the version is not specified, supporting any version of the feature causes the method to return true.

Example:

```
aNodeObject.isSupported("Core", "2.0")
```

Document

Lets you work with documents. This section includes the following topics:

- ♦ “doctype” on page 274

- ♦ “documentElement” on page 274
- ♦ “implementation” on page 274
- ♦ “text” on page 274
- ♦ “createAttribute(name)” on page 275
- ♦ “createCDATASection(data)” on page 275
- ♦ “createComment(data)” on page 275
- ♦ “createDocumentFragment()” on page 275
- ♦ “createElement(tagName)” on page 275
- ♦ “createEntityReference(name)” on page 275
- ♦ “createProcessingInstruction(target,data)” on page 275
- ♦ “createTextNode(data)” on page 275
- ♦ “getElementsByTagName(tagName)” on page 275
- ♦ “reset()” on page 276
- ♦ “setDTD(Node RootElementName, Object PublicName, Object URL)” on page 276
- ♦ “setValue(Object aValue)” on page 276
- ♦ “toString()” on page 276
- ♦ “XPath(String asPattern)” on page 276
- ♦ “importNode(sourceNode, deep)” on page 276
- ♦ “createElementNS(namespaceURI, qualifiedName)” on page 277
- ♦ “createAttributeNS(namespaceURI, qualifiedName)” on page 277
- ♦ “getElementsByTagNameNS(namespaceURI, localName)” on page 277
- ♦ “getElementById(elementId)” on page 278
- ♦ “setSkipNameSpaces(abFlag)” on page 278
- ♦ “setEncoding(encoding)” on page 278

doctype

W3C DOM Level 1 Document property. This property returns a DocumentType object reflecting the DTD for the document. A Document also has all the properties and methods of Node.

documentElement

W3C DOM Level 1 Document property. This property returns an Element object (the root element). A Document also has all the properties and methods of Node.

implementation

W3C DOM Level 1 Document property. This property returns a DOMImplementation object. A Document also has all the properties and methods of Node.

text

Designer extension property. This property returns a concatenated string of all the text nodes (content) under it.

createAttribute(name)

`Attr createAttribute(name)`

W3C DOM Level 1 Document method. This method returns an unattached `Attr` object. The `name` parameter is of type `String`. A `Document` also has all the properties and methods of `Node`.

createCDATASection(data)

`CDATASection createCDATASection(data)`

W3C DOM Level 1 Document method. This method returns an unattached `CDATASection` object. The `data` parameter is of type `String`. A `Document` also has all the properties and methods of `Node`.

createComment(data)

`Comment createComment(data)`

W3C DOM Level 1 Document method. This method returns an unattached `Comment` object. The `data` parameter is of type `String`. A `Document` also has all the properties and methods of `Node`.

createDocumentFragment()

`DocumentFragment createDocumentFragment()`

W3C DOM Level 1 Document method. This method returns an unattached `DocumentFragment`. A `Document` also has all the properties and methods of `Node`.

createElement(tagName)

`Element createElement(tagName)`

W3C DOM Level 1 Document method. This method creates an unattached `Element`. The `tagName` parameter is of type `String`. A `Document` also has all the properties and methods of `Node`.

createEntityReference(name)

`EntityReference createEntityReference(name)`

W3C DOM Level 1 Document method. Creates an unattached `EntityReference`. The `name` parameter is of type `String`. A `Document` also has all the properties and methods of `Node`.

createProcessingInstruction(target,data)

`ProcessingInstruction createProcessingInstruction(target,data)`

W3C DOM Level 1 Document method. This method returns an unattached `ProcessingInstruction` object. The `target` and `data` parameters are of type `String`. A `Document` also has all the properties and methods of `Node`.

createTextNode(data)

`Text createTextNode(data)`

W3C DOM Level 1 Document method. This method creates an unattached `Text` object. The `data` parameter is of type `String`. A `Document` also has all the properties and methods of `Node`.

getElementsByTagName(tagName)

`NodeList getElementsByTagName(tagName)`

W3C DOM Level 1 Document method. This method returns a NodeList object consisting of the tagname element nodes. The tagName parameter is of type String. A Document also has all the properties and methods of Node.

reset()

`void reset()`

W3C DOM Level 1 Document method. Clears the document.

setDTD(Node RootElementName, Object PublicName, Object URL)

`setDTD(Node RootElementName, Object PublicName, Object URL)`

ECMAScript extension method. Sets DTD file for the document.

setValue(Object aValue)

`setValue(Object aValue)`

ECMAScript extension method. Sets the Value of a document from the passed objects. If the passed object is another document, then this method copies child nodes (elements and attributes). If the passed object is text, the text is parsed to create a DOM.

toString()

`String toString()`

ECMAScript extension method. Converts a DOM document to an XML formatted string.

Example:

`Input.XPath("root/child").item(0).toString()`

XPath(String asPattern)

`NodeList XPath(XPathType asPattern)`

ECMAScript extension method. XPathTypes can be of type NodeList, String, Number, or Boolean. Usually used to return a Nodelist matching the XPath pattern. Use brackets to select a particular node from the list. For example, `Input.XPath("INVOICE/LINEITEM[1]")` or `Input.XPath("INVOICE/LINEITEM[last()]")`. Use the @ symbol to select a node by attribute. For example, `Input.XPath("INVOICE/LINEITEM[@myattr]")` To select by attribute value: `Input.XPath("INVOICE/LINEITEM[@myattr='abc']")`.

importNode(sourceNode, deep)

`Node importNode(sourceNode, deep)`

W3C DOM Level 2 Document method. Imports a node from a document to the current document. This method creates a new copy of the sourceNode. The sourceNode is not altered. A Document also has all the properties and methods of Node.

Table 10-13 Parameters for ImportNode Method

Parameter	Description
sourceNode	The node to import.

Parameter	Description
deep	A boolean. If true, recursively import the subtree under the specified node. If false, import only the node itself.

Example:

```
Temp.importNode(Input.XPath("A/B[2]"), false)
```

createElementNS(namespaceURI, qualifiedName)

```
Element createElementNS(namespaceURI, qualifiedName)
```

W3C DOM Level 2 Document method. Creates an Element of the given qualifiedName and namespaceURI. A Document also has all the properties and methods of Node.

Table 10-14 Parameters for createElementNS Method

Parameter	Description
namespaceURI	A string representing the namespace URI that you want to create for the element.
qualifiedName	A string representing the name to create for the element. Note: qualifiedName = namespaceprefix + : + localName

Example:

```
Temp.createElementNS("someURI", "namespace:PRICE")
```

createAttributeNS(namespaceURI, qualifiedName)

```
Attr createAttributeNS(namespaceURI, qualifiedName)
```

W3C DOM Level 2 Document method. Creates an Attribute of the given qualifiedName and namespaceURI. A Document also has all the properties and methods of Node.

Table 10-15 Parameters for createAttributeNS Method

Parameter	Description
namespaceURI	A string representing the namespace URI that you want to create for the attribute.
qualifiedName	A string representing the name to create for the attribute. Note: qualifiedName = namespaceprefix + : + localName

Example:

```
Temp.createAttributeNS("someURI", "namespace:PRICE")
```

getElementsByTagNameNS(namespaceURI, localName)

```
NodeList getElementsByTagNameNS(namespaceURI, localName)
```

W3C DOM Level 2 Document method. Returns a NodeList of all the Elements with a given localName and namespace URI, in the order in which they are encountered in a preorder traversal of the Document tree. A Document also has all the properties and methods of Node.

Table 10-16 Parameters for *getElementsByTagNameNS* Method

Parameter	Description
namespaceURI	A string of the elements on which to match. The special value "*" matches all namespaces.
qualifiedName	A string of the elements on which to match. The special value "*" matches all local names.

Example:

```
Temp.getElementsByTagNameNS("someURI", "someName")
```

getElementById(elementId)

```
Element getElementById(elementId)
```

W3C DOM Level 2 Document method. Returns the Element the ID of which is given by `elementId`. If no such element exists, returns null. Behavior is not defined if more than one element has this ID. A Document also has all the properties and methods of Node.

Example;

```
Temp.getElementById("someId")
```

setSkipNameSpaces(abFlag)

```
void setSkipNameSpaces(boolean flag)
```

This method can be used to turn off usage of namespaces and match nodes without any prefixes, behaving like a wildcard match.

setEncoding(encoding)

```
void setEncoding(String encoding)
```

This method sets the character set encoding for the document.

Element

Lets you work with elements. This section includes the following topics:

- ♦ [“tagName” on page 279](#)
- ♦ [“text” on page 279](#)
- ♦ [“booleanValue\(\)” on page 279](#)
- ♦ [“countOfElement\(String propertyName\)” on page 279](#)
- ♦ [“doubleValue\(\)” on page 279](#)
- ♦ [“exists\(String propertyName\)” on page 280](#)
- ♦ [“getAttribute\(name\)” on page 280](#)
- ♦ [“getAttributeNode\(name\)” on page 280](#)
- ♦ [“getElementsByTagName\(name\)” on page 280](#)
- ♦ [“getIndex\(\)” on page 280](#)
- ♦ [“getParent\(\)” on page 280](#)

- ♦ “normalize()” on page 280
- ♦ “removeAttribute(name)” on page 280
- ♦ “removeAttributeNode(oldAttr)” on page 280
- ♦ “setAttribute(name,value)” on page 280
- ♦ “setAttributeNode(newAttr)” on page 281
- ♦ “setIndex(int aiIndex)” on page 281
- ♦ “setText(String asText)” on page 281
- ♦ “setValue(Object aValue)” on page 281
- ♦ “toNumber()” on page 281
- ♦ “toString()” on page 281
- ♦ “XPath(XPathType asPattern)” on page 281
- ♦ “getAttributeNS(namespaceURI, localName)” on page 281
- ♦ “setAttributeNS(namespaceURI, qualifiedName, value)” on page 282
- ♦ “removeAttributeNS(namespaceURI, localName)” on page 282
- ♦ “getAttributeNodeNS(namespaceURI, localName)” on page 283
- ♦ “setAttributeNodeNS(newAttr)” on page 283
- ♦ “getElementsByTagNameNS(namespaceURI, localName)” on page 283
- ♦ “hasAttribute(name)” on page 284
- ♦ “hasAttributeNS(namespaceURI, localName)” on page 284

tagName

W3C DOM Level 1 Element property. This property returns a String object containing the element name. An Element also has all the properties and methods of Node.

text

Designer extension property. This property returns the concatenated text of all the text nodes under it.

booleanValue()

`boolean booleanValue()`

ECMAScript extension method. Returns the boolean value (true or false) of this object, if possible.

countOfElement(String propertyName)

`Number countOfElement(String propertyName)`

ECMAScript extension method. Returns a count of the named child.

doubleValue()

`double doubleValue()`

ECMAScript extension method. Returns a double value for this object if possible.

exists(String propertyName)

`Boolean exists(String propertyName)`

ECMAScript extension method. Checks for the existence of the named child.

getAttribute(name)

`String getAttribute(name)`

W3C DOM Level 1 Element method. This method returns a String consisting of the attribute value. The name parameter is of type String. An Element also has all the properties and methods of Node.

getAttributeNode(name)

`Attr getAttributeNode(name)`

W3C DOM Level 1 Element method. This method returns an Attr. The name parameter is of type String. An Element also has all the properties and methods of Node.

getElementsByTagName(name)

`NodeList getElementsByTagName(name)`

W3C DOM Level 1 Element method. Returns a NodeList of all elements with a specified name. The name parameter is of type String. An Element also has all the properties and methods of Node.

getIndex()

`int getIndex()`

ECMAScript extension method. Returns the current index.

getParent()

`Node getParent()`

ECMAScript extension method. Returns the parent element.

normalize()

`void normalize()`

W3C DOM Level 1 Element method. This method returns a void. An Element also has all the properties and methods of Node.

removeAttribute(name)

`void removeAttribute(name)`

W3C DOM Level 1 Element method. This method removes an attribute from an element. The name parameter is of type String. An Element also has all the properties and methods of Node.

removeAttributeNode(oldAttr)

`Attr removeAttributeNode(oldAttr)`

W3C DOM Level 1 Element method. This method removes an attribute from an element and returns an unattached Attr. The oldAttr parameter is of type Attr. An Element also has all the properties and methods of Node.

setAttribute(name,value)

`void setAttribute(name, value)`

W3C DOM Level 1 Element method. This method sets the value of an attribute node for an element. The `name` parameter is of type `String`. The `value` parameter is of type `String`. An `Element` also has all the properties and methods of `Node`.

setAttributeNode(newAttr)

`Attr setAttributeNode (newAttr)`

W3C DOM Level 1 Element method. This method attaches an attribute node to an element. The `newAttr` parameter is of type `Attr`. An `Element` also has all the properties and methods of `Node`.

setIndex(int aiIndex)

`setIndex (int aiIndex)`

ECMAScript extension method. Sets the iterator index value for this element.

setText(String asText)

`setText (String asText)`

ECMAScript extension method. Sets the text node associated with this element.

setValue(Object aValue)

`setValue (Object aValue)`

ECMAScript extension method. Sets the value of an element from the passed object. If the passed object is another element, then this method also copies child nodes (elements and attributes).

toNumber()

`Number toNumber()`

ECMAScript extension method. Gets the text node and converts it to a number.

toString()

`String toString()`

ECMAScript extension method. Gets the text node associated with this element.

XPath(XPathType asPattern)

`NodeList XPath (XPathType asPattern)`

ECMAScript extension method. The `XPathType` parameter can be of type `NodeList`, `String`, `Number`, or `Boolean`. Usually used to return a `Nodelist` matching the `XPath` pattern. Use brackets to select a particular node from the list. For example, `Input.XPath ("INVOICE/LINEITEM[1]")` or `Input.XPath ("INVOICE/LINEITEM[last()]")`. Use the `@` symbol to select a node by attribute. For example, `Input.XPath ("INVOICE/LINEITEM[@myattr]")`. To select by attribute value: `Input.XPath ("INVOICE/LINEITEM[@myattr='abc']")`.

getAttributeNS(namespaceURI, localName)

`string getAttributeNS (namespaceURI, localName)`

W3C DOM Level 2 Element method. Returns the `Attr` value as a string. An `Element` also has all the properties and methods of `Node`.

Table 10-17 *Parameters for getAttributeNS Method*

Parameter	Description
namespaceURI	Specifies a string representing the namespace URI of the target Attr.
localName	Specifies a string of the localName of the target Attr.

Example:

```
Temp.XPath("A/B[0]").getAttributeNS("someURI", "someAttr")
```

setAttributeNS(namespaceURI, qualifiedName, value)

```
void setAttributeNS(namespaceURI, qualifiedName, value)
```

W3C DOM Level 2 Element method. Adds a new attribute. If an attribute with the same namespaceURI and localName is already present in the element, its prefix is changed to be the prefix part of the qualifiedName parameter, and its value is changed to be the value parameter. An Element also has all the properties and methods of Node.

Table 10-18 *Parameters for setAttributeNS Method*

Parameter	Description
namespaceURI	The namespace URI of tthe attribute to create or alter.
qualifiedName	Specifies the qualified name of the attribute to create or alter.
	TIP: qualifiedName = namespaceprefix + : + localName
value	Specifies the value to set in string form.

Example:

```
Temp.XPath("A/B[0]").setAttributeNS("someURI", "someAttrName",  
"someAttrvalue")
```

removeAttributeNS(namespaceURI, localName)

```
void removeAttributeNS(namespaceURI, localName)
```

W3C DOM Level 2 Element method. Removes an attribute by local name and namespace URI. If the removed attribute has a default value, it is immediately replaced. The replacing attribute has the same namespace URI and local name, as well as the original prefix. An Element also has all the properties and methods of Node.

Table 10-19 *Parameters for removeAttributeNS Method*

Parameter	Description
namespaceURI	Specifies the namespaceURI of the attribute to remove.
localName	Specifies the name of the attribute to remove.

Example:

```
Temp.XPath("A/B[0]").removeAttributeNS("someURI", "someAttrName")
```

getAttributeNodeNS(namespaceURI, localName)

```
Attr getAttributeNodeNS(namespaceURI, localName)
```

W3C DOM Level 2 Element method. Retrieves an attribute node by local name and namespace URI. An Element also has all the properties and methods of Node.

Table 10-20 Parameters for *getAttributeNodeNS* Method

Parameter	Description
namespaceURI	Specifies the namespaceURI of the attribute to retrieve.
localName	Specifies the name of the attribute to retrieve.

Example:

```
Temp.XPath("A/B[0]").getAttributeNodeNS("someURI", "someAttr")
```

setAttributeNodeNS(newAttr)

```
Attr setAttributeNodeNS(newAttr)
```

W3C DOM Level 2 Element method. Adds a new attribute. If an attribute with the same local name and namespace URI is already present in the element, it is replaced by the new attribute. If the newAttr attribute replaces an existing attribute with the same local name and namespace URI, the replaced Attr node is returned, otherwise null is returned. The newAttr parameter is a new attribute object. An Element also has all the properties and methods of Node.

Example:

```
Temp.XPath("A/B[0]").setAttributeNodeNS(newAttr)
```

getElementsByTagNameNS(namespaceURI, localName)

```
NodeList getElementsByTagNameNS(namespaceURI, localName)
```

W3C DOM Level 2 Element method. Returns a NodeList of all the descendant Elements with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of this Element tree. An Element also has all the properties and methods of Node.

Table 10-21 Parameters for *getElementsByTagNameNS* Method

Parameter	Description
namespaceURI	Specifies the namespaceURI of the elements on which to match. The special value "*" matches all namespaces.
localName	Specifies the localName of the elements on which to match. The special value "*" matches all local names.

Example:

```
Temp.XPath("A/B[0]").getElementsByTagNameNS("someURI", "someName")
```

hasAttribute(name)

`boolean hasAttribute()`

W3C DOM Level 2 Element method. Returns true when an attribute with a given name is specified for this element or has a default value. Otherwise, returns false. The parameter `name` is a string that specifies the attribute name for which to look. An Element also has all the properties and methods of Node.

Example:

```
Temp.XPath("A/B[0]").hasAttribute("someName")
```

hasAttributeNS(namespaceURI, localName)

`boolean hasAttributeNS(namespaceURI, localName)`

W3C DOM Level 2 Element method. Returns true when an attribute with a given local name and namespace URI is specified on this element or has a default value. Otherwise, returns false. An Element also has all the properties and methods of Node.

Table 10-22 Parameters for *hasAttributeNS* Method

Parameter	Description
<code>namespaceURI</code>	Specifies the namespaceURI of the attribute for which to look.
<code>localName</code>	Specifies the localName of the attribute for which to look.

Example:

```
Temp.XPath("A/B[0]").hasAttributeNS("someURI", "someName")
```

Attribute

Lets you work with attributes. This section includes the following topics:

- ♦ [“name” on page 284](#)
- ♦ [“specified” on page 284](#)
- ♦ [“text” on page 285](#)
- ♦ [“value” on page 285](#)
- ♦ [“setValue\(Object aValue\)” on page 285](#)
- ♦ [“toString\(\)” on page 285](#)
- ♦ [“ownerElement” on page 285](#)

name

W3C DOM Level 1 attribute property. This property returns a String object indicating the tag name of the attribute. An attribute also has all the properties and methods of Node.

specified

W3C DOM Level 1 Attr property. This property returns a boolean. An attribute also has all the properties and methods of Node.

text

Designer extension property. This property returns the text value of the attribute.

value

W3C DOM Level 1 Attr property. This property returns a String object representing the text value of the attribute. An attribute also has all the properties and methods of Node.

setValue(Object aValue)

`setValue(Object aValue)`

Designer extension method. Sets the value of an Attribute from the passed object.

toString()

`String toString()`

ECMAScript extension method. Gets the text node associated with the attribute.

ownerElement

W3C DOM Level 2 Attr property. Returns the Element node to which this attribute is attached. Returns null if this attribute is not in use. An Attr also has all the properties and methods of Node.

Example:

`attributeObject.ownerElement`

CharacterData

Lets you work with character data. This section includes the following topics:

- ♦ [“data” on page 285](#)
- ♦ [“length” on page 285](#)
- ♦ [“appendData\(arg\)” on page 285](#)
- ♦ [“insertData\(offset, arg\)” on page 286](#)
- ♦ [“deleteData\(offset, count\)” on page 286](#)
- ♦ [“replaceData\(offset, count, arg\)” on page 286](#)
- ♦ [“substringData\(offset, count\)” on page 286](#)

data

W3C DOM Level 1 CharacterData property. This property is of type String and represents the contents of the CharacterData object. CharacterData also has all the properties and methods of Node.

length

W3C DOM Level 1 CharacterData property. This property represents the length of the CharacterData object. CharacterData also has all the properties and methods of Node.

appendData(arg)

`void appendData(arg)`

W3C DOM Level 1 `CharacterData` method. This method appends text to the `CharacterData` object. The `arg` parameter is of type `String`. `CharacterData` also has all the properties and methods of `Node`.

`insertData(offset, arg)`

`void insertData(offset, arg)`

W3C DOM Level 1 `CharacterData` method. This method inserts text in the `CharacterData` object. The `offset` parameter is of type unsigned long. The `arg` parameter is of type `String`. `CharacterData` also has all the properties and methods of `Node`.

`deleteData(offset, count)`

`void deleteData(offset, count)`

W3C DOM Level 1 `CharacterData` method. This method deletes text in the `CharacterData` object. The `offset` and `count` parameters are of type unsigned long. `CharacterData` also has all the properties and methods of `Node`.

`replaceData(offset, count, arg)`

`void replaceData(offset, count, arg)`

W3C DOM Level 1 `CharacterData` method. This method replaces text in the `CharacterData` object. The `offset` and `count` parameters are of type unsigned long. The `arg` parameter is of type `String`. `CharacterData` also has all the properties and methods of `Node`.

`substringData(offset, count)`

`String substringData(offset, count)`

W3C DOM Level 1 `CharacterData` method. This method returns a substring of the `CharacterData` object. The `offset` and `count` parameters are of type unsigned long. `CharacterData` also has all the properties and methods of `Node`.

NodeList

Lets you work with node lists. This section includes the following topics:

- ♦ [“length” on page 286](#)
- ♦ [“avg\('\[NodeList\]’\)” on page 287](#)
- ♦ [“count\('\[NodeList\]’\)” on page 287](#)
- ♦ [“item\(index\)” on page 287](#)
- ♦ [“min\('\[NodeList\]’\)” on page 287](#)
- ♦ [“max\('\[NodeList\]’\)” on page 287](#)
- ♦ [“sum\('\[NodeList\]’\)” on page 288](#)
- ♦ [“where\(XPathType asPattern\)” on page 288](#)
- ♦ [“toNumber\(\)” on page 288](#)

length

W3C DOM Level 1 `NodeList` property. This property returns the number of nodes in a `NodeList` object.

avg(['NodeList'])

Number avg(['NodeList'])

ECMAScript aggregate extension method. Returns a number equal to the average value in the NodeList. The NodeList parameter of type XPath. If no parameter is supplied, then the current NodeList/GroupName is used. The function argument should be in single quotes, and must be escaped in the case of nested calls.

Example:

```
Input.XPath("rootElem/childElem").avg()
```

count(['NodeList'])

Number count(['NodeList'])

ECMAScript aggregate extension method. Returns a number equal to a count of the nodes in the NodeList that have data. Nodes without data, or nodes with only child elements will not be counted. To count all nodes, use the .length property on a nodeList object. The optional NodeList parameter is of type XPath. If no parameter is supplied (the usual case), then the current NodeList/GroupName is used. The function argument should be in single quotes, and must be escaped in the case of nested calls.

Example:

```
Input.XPath("rootElem/childElem").count()
```

item(index)

Node item(index)

W3C DOM Level 1 NodeList method. This method returns the indicated Node from the NodeList. The index parameter is of type unsigned long. The Index is 0-based.

min(['NodeList'])

Number min(['NodeList'])

ECMAScript aggregate extension method. Returns a number equal to the lowest value in the NodeList. The NodeList parameter of type XPath. If no parameter is supplied, then the current NodeList/GroupName is used. The function argument should be in single quotes, and must be escaped in the case of nested calls.

Example:

```
Input.XPath("rootElem/childElem").min()
```

max(['NodeList'])

Number max(['NodeList'])

ECMAScript aggregate extension method. Returns a number equal to the highest value in the NodeList. The NodeList parameter of type XPath. If no parameter is supplied, then the current NodeList/GroupName is used. The function argument should be in single quotes, and must be escaped in the case of nested calls.

Example:

```
Input.XPath("rootElem/childElem").max()
```

sum("[NodeList"])

Number sum(' [NodeList] ')

ECMAScript aggregate extension method. Returns a number equal to the sum of the values in NodeList. The `NodeList` parameter of type XPath. If no parameter is supplied, then the current `NodeList/GroupName` is used. The function argument should be in single quotes, and must be escaped in the case of nested calls.

Example:

```
Input.XPath("rootElem/childElem").sum()
```

where(XPathType asPattern)

NodeList where(String asPattern)

ECMAScript extension method. Gets a `NodeList` of nodes matching the XPath pattern.

toNumber()

toNumber()

Converts the data of the first instance in the `NodeList` to an ECMAScript Number object. Any alphabetic characters or embedded spaces in data returns NaN. Leading and trailing spaces are permitted.

Example:

```
var myNum = Input.XPath("Invoice/Amount").toNumber()
```

NamedNodeMap

Lets you work with named node maps. This section includes the following topics:

- ♦ [“length” on page 288](#)
- ♦ [“getNamedItem\(name\)” on page 288](#)
- ♦ [“getNamedItemNS\(namespaceURI, localName\)” on page 289](#)
- ♦ [“item\(index\)” on page 289](#)
- ♦ [“removeNamedItem\(name\)” on page 289](#)
- ♦ [“removeNamedItemNS\(namespaceURI, localName\)” on page 289](#)
- ♦ [“setNamedItem\(arg\)” on page 289](#)
- ♦ [“setNamedItemNS\(Node arg\)” on page 290](#)

length

`length` W3C DOM Level 1 `NamedNodeMap` property. This property returns the number of nodes in a `NamedNodeMap`.

getNamedItem(name)

Node getNamedItem(name)

W3C DOM Level 1 `NamedNodeMap` method. This method returns all selected Nodes of the indicated name. The name parameter is of type `String`.

getNamedItemNS(namespaceURI, localName)

Node getNamedItemNS(namespaceURI, localName)

W3C DOM Level 2 NamedNodeMap method. Returns a node specified by local name and namespace URI.

Table 10-23 Parameters for NamedNodeMap Method

Parameter	Description
namespaceURI	Specifies the namespaceURI of the node to retrieve.
localName	Specifies the localName of the node to retrieve.

Example:

```
Temp.XPath("A/B").item(0).getAttributes().getNamedItemNS("someURI",  
"anAttrName")
```

item(index)

Node item(index)

W3C DOM Level 1 NamedNodeMap method. This method returns the indicated Node from the NamedNodeMap. The `index` parameter is of type unsigned long. The index is 0-based.

removeNamedItem(name)

Node removeNamedItem(name)

W3C DOM Level 1 NamedNodeMap method. This method removes the indicated node from the NamedNodeMap and returns an unattached node. The `name` parameter is of type String.

removeNamedItemNS(namespaceURI, localName)

Node removeNamedItemNS(namespaceURI, localName)

W3C DOM Level 2 NamedNodeMap method. Removes and returns the node specified by namespace URI and local name.

Table 10-24 Parameters for removeNamedItemNS Method

Parameter	Description
namespaceURI	Specifies the namespaceURI of the node to remove.
localName	Specifies the localName of the node to remove.

Example:

```
Temp.XPath("A/B").item(0).getAttributes().  
removeNamedItemNS("someURI", "anAttrName")
```

setNamedItem(arg)

Node setNamedItem(arg)

W3C DOM Level 1 NamedNodeMap method. This method returns a Node. The `arg` parameter is of type Node.

setNamedItemNS(Node arg)

`Node setNamedItemNS (arg)`

W3C DOM Level 2 NamedNodeMap method. If the new Node replaces an existing node, the replaced Node is returned, otherwise null is returned.

Example:

```
var item = Temp.XPath("A/B").item(0);  
item.getAttributes().setNamedItemNS(aNodeObject)
```

Text

Lets you work with text.

splitText(offset)

`Text splitText (offset)`

W3C DOM Level 1 Element method. This method removes the text up to the offset and creates an unattached text node with the removed text. The `offset` parameter is of type unsigned long. A Text also has all the properties and methods of CharacterData.

DocumentType

Lets you work with document types. This section includes the following topics:

- ♦ [“name” on page 290](#)
- ♦ [“entities” on page 290](#)
- ♦ [“internalSubset” on page 290](#)
- ♦ [“notations” on page 291](#)
- ♦ [“publicId” on page 291](#)
- ♦ [“systemId” on page 291](#)

name

W3C DOM Level 1 DocumentType property. This property returns a String representing the document type name.

entities

W3C DOM Level 1 DocumentType property. This property returns a NamedNodeMap of the entities defined in the document.

internalSubset

W3C DOM Level 2 DocumentType property. This property returns a String representing the internal subset as a string.

notations

W3C DOM Level 1 `DocumentType` property. This property returns a `NamedNodeMap` of the notations defined in the document.

publicId

W3C DOM Level 2 `DocumentType` property. This property returns a `String` representing the public identifier of the external subset.

systemId

W3C DOM Level 2 `DocumentType` property. This property returns a `String` representing the system identifier of the external subset.

DOMImplementation

Lets you work with DOM implementations. This section includes the following topics:

- ♦ “`createDocument(namespaceURI, qualifiedName, doctype)`” on page 291
- ♦ “`createDocumentType(qualifiedName, publicID, systemID)`” on page 291
- ♦ “`hasFeature(feature, version)`” on page 292

`createDocument(namespaceURI, qualifiedName, doctype)`

```
Document createDocument(namespaceURI, qualifiedName, doctype)
```

W3C DOM Level 2 `DOMImplementation` method. Creates an XML Document object of the specified type with its document element.

Table 10-25 Parameters for *DOMImplementation* Method

Parameter	Description
<code>namespaceURI</code>	Specifies the <code>namespaceURI</code> of the document element to create.
<code>qualifiedName</code>	Specifies the qualified name of the document element to create. <code>qualifiedName = namespaceprefix + : + localName</code>
<code>doctypei</code>	Specifies the type of document to create, or null.

`createDocumentType(qualifiedName, publicID, systemID)`

```
DocumentType createDocumentType(qualifiedName, publicID, systemID)
```

W3C DOM Level 2 `DOMImplementation` method. Creates an empty `DocumentType` node.

Parameters: `qualifiedName`—is a string of the name of the document type to create. `publicID` is the external subset public identifier. `systemID` is the external subset system identifier. Note:

`qualifiedName = namespaceprefix + : + localName`

Table 10-26 *Parameters for createDocumentType Method*

Parameter	Description
qualifiedName	Specifies the qualified name of the document element to create. qualifiedName = namespaceprefix + : + localName
publicID	Specifies the external subset public identifier.
systemID	Specifies the external subset system identifier.

hasFeature(feature, version)

boolean hasFeature(feature, version)

W3C DOM Level 1 DOMImplementation method. This method returns a boolean. The `feature` parameter is of type String. The `version` parameter is of type String.

Notation

Lets you work with notation. This section includes the following topics:

- ♦ [“publicId” on page 292](#)
- ♦ [“systemId” on page 292](#)

publicId

W3C DOM Level 2 This property returns a String representing the public identifier of the external subset.

systemId

W3C DOM Level 2 property. This property returns a String representing the system identifier of the external subset.

Entity

Lets you work with entities. This section includes the following topics:

- ♦ [“publicId” on page 292](#)
- ♦ [“systemId” on page 292](#)
- ♦ [“notationName” on page 293](#)

publicId

W3C DOM Level 2 property. This property returns a String representing the public identifier of the external subset.

systemId

W3C DOM Level 2 property. This property returns a String representing the system identifier of the external subset.

notationName

W3C DOM Level 1 Entity property. This property is of type String. An Entity also has all the properties and methods of Node.

ProcessingInstruction

Lets you work with processing instructions. This section includes the following topics:

- ♦ “target” on page 293
- ♦ “data” on page 293

target

W3C DOM Level 1 ProcessingInstruction property. This property is a String representation of the target part of a Processing Instruction.

data

W3C DOM Level 1 ProcessingInstruction property. This property is a String representation of the data part of a Processing Instruction.

10.3.3 ECMAScript Core

This section lists all ECMAScript core methods and properties supported by the ECMA expression builder. This section includes the following topics:

- ♦ “Array Object” on page 293
- ♦ “Boolean Object” on page 294
- ♦ “Date Object” on page 295
- ♦ “Function Object” on page 300
- ♦ “Global” on page 301
- ♦ “Math Object” on page 302
- ♦ “Number Object” on page 307
- ♦ “Object” on page 308
- ♦ “String Object” on page 309

Array Object

Lets you work with arrays. This section includes the following topics:

- ♦ “Array(item0, item1, . . .)” on page 294
- ♦ “join(separator)” on page 294
- ♦ “length” on page 294
- ♦ “reverse()” on page 294
- ♦ “sort(comparefn)” on page 294
- ♦ “toString()” on page 294

Array(item0, item1, . . .)

`Array()`

Constructor

join(separator)

`Array join(separator)`

The elements of the array are converted to strings, and these strings are then concatenated, separated by occurrences of the separator. If no separator is provided, a single comma is used as the separator.

length

Array length. The length property of this Array object

reverse()

`reverse()`

The elements of the array are rearranged so as to reverse their order. The operation is done in-place, meaning that the original array is modified.

sort(comparefn)

`Array sort()`

The elements of this array are sorted. The sort is not necessarily stable. If `comparefn` is supplied, it should be a function that accepts two arguments `x` and `y` and returns a negative value if `x < y`, zero if `x = y`, or a positive value if `x > y`.

toString()

`Array toString()`

The elements of this object are converted to strings, and these strings are then concatenated, separated by comma characters. The result is the same as if the built-in `join` method were invoked for this object with no argument.

Boolean Object

There is seldom a need to use the object version of Boolean in place of `true/false` literal values. This object is provided for completeness. It is specified in ECMA-262.

This section includes the following topics:

- ♦ [“Boolean\(\)” on page 294](#)
- ♦ [“toString\(\)” on page 294](#)
- ♦ [“valueOf\(\)” on page 295](#)

Boolean()

`Boolean([true/false])`

Constructor. Optionally takes one of `true` or `false` as an argument.

toString()

`Boolean toString()`

If this Boolean value is true, then the string “true” is returned. Otherwise, this Boolean value must be false, and the string “false” is returned.

`valueOf()`

`Boolean valueOf()`

Returns this Boolean value.

Date Object

Lets you work with dates and times. This section includes the following topics:

- ♦ “Date()” on page 296
- ♦ “getDate()” on page 296
- ♦ “getDay()” on page 296
- ♦ “getFullYear()” on page 296
- ♦ “getHours()” on page 296
- ♦ “getMilliseconds()” on page 296
- ♦ “getMinutes()” on page 296
- ♦ “getMonth()” on page 297
- ♦ “getSeconds()” on page 297
- ♦ “getTime()” on page 297
- ♦ “getTimezoneOffset()” on page 297
- ♦ “getUTCDate()” on page 297
- ♦ “getUTCDay()” on page 297
- ♦ “getUTCFullYear()” on page 297
- ♦ “getUTCHours()” on page 297
- ♦ “getUTCMilliseconds()” on page 297
- ♦ “getUTCMinutes()” on page 297
- ♦ “getUTCSeconds()” on page 298
- ♦ “getYear()” on page 298
- ♦ “parse(string)” on page 298
- ♦ “setDate(date)” on page 298
- ♦ “setFullYear(year[,mon[,date]])” on page 298
- ♦ “setHours(hour[,min[,sec[,ms]])” on page 298
- ♦ “setMilliseconds(ms)” on page 298
- ♦ “setMinutes(min[,sec[,ms]])” on page 298
- ♦ “setMonth(mon[,date])” on page 298
- ♦ “setSeconds(sec [, ms])” on page 299
- ♦ “setTime(time)” on page 299
- ♦ “setUTCDate(date)” on page 299
- ♦ “setUTCFullYear(year[,mon[,date]])” on page 299

- ♦ “setUTCHours(min[,sec[,ms]])” on page 299
- ♦ “setUTCMilliseconds(ms)” on page 299
- ♦ “setUTCMinutes(min[,sec[,ms]])” on page 299
- ♦ “setUTCMonth(mon[,date])” on page 299
- ♦ “setUTCSeconds(sec [, ms])” on page 300
- ♦ “setYear(year)” on page 300
- ♦ “toLocaleString()” on page 300
- ♦ “toString()” on page 300
- ♦ “toUTCString()” on page 300
- ♦ “UTC()” on page 300
- ♦ “valueOf()” on page 300

Date()

Date ()

The constructor of the Date can have various signatures. The date constructor format can accept up to 7 parameters. Here is the format: new Date(year,month,date,hrs,mins,secs,ms)

getDate()

getDate ()

Returns DateFromTime(LocalTime(t)).

getDay()

getDay ()

Returns WeekDay(LocalTime(t)). The days of week are numbered from 0-6. The number 0 represents Sunday and 6 represents Saturday.

getFullYear()

getFullYear ()

Returns YearFromTime(LocalTime(t)).

getHours()

getHours ()

Returns HourFromTime(LocalTime(t)).

getMilliseconds()

getMilliseconds ()

Returns msFromTime(LocalTime(t)).

getMinutes()

getMinutes ()

Returns MinFromTime(LocalTime(t)).

getMonth()

`getMonth()`

Returns `MonthFromTime(LocalTime(t))`. The months are returned as an integer value from 0-11. The number 0 represents January and 11 represents December.

getSeconds()

`getSeconds()`

Returns `SecFromTime(LocalTime(t))`.

getTime()

`getTime()`

Returns a number, which is this time value. The number value is a millisecond representation of the specified Date object.

getTimezoneOffset()

`getTimezoneOffset()`

Returns $(t * \text{LocalTime}(t)) / \text{msPerMinute}$. The difference is in minutes between (GMT) and local time.

getUTCDate()

`getUTCDate()`

Returns `DateFromTime(t)`.

getUTCDay()

`getUTCDay()`

Returns `WeekDay(t)`. The days of week are numbered from 0-6. The number 0 represents Sunday and 6 represents Saturday.

getUTCFullYear()

`getUTCFullYear()`

Returns `YearFromTime(t)`. There is no `getYearUTC` method, so this method must be used to obtain a year from an UTC Date object.

getUTCHours()

`getUTCHours()`

Returns `HourFromTime(t)`.

getUTCMilliseconds()

`getUTCMilliseconds()`

Returns `msFromTime(t)`.

getUTCMinutes()

`getUTCMinutes()`

Returns `MinFromTime(t)`.

getUTCSeconds()

`getUTCSeconds()`

Returns `SecFromTime(t)`.

getFullYear()

`getFullYear()`

Returns `YearFromTime(LocalTime(t))`—1900. The function `getFullYear()` is preferred for nearly all purposes because it avoids the year 2000 problem.

parse(string)

`parse(string)`

Applies the `ToString` operator to its argument and interprets the resulting string as a date; it returns a number, the UTC time value corresponding to the date. The string is interpreted as a local time, a UTC time, or a time in some other time zone, depending on the contents of the string.

setDate(date)

`setDate(date)`

Sets the day of the month, using an integer from 1 to 31, for the supplied date according to local time.

setFullYear(year[,mon[,date]])

`setFullYear(year[,mon[,date]])`

Sets the `[Value]` property of this value to UTC `ECMAScript.Date`. Returns the value of the `[Value]` property of this value.

setHours(hour[,min[,sec[,ms]]])

`setHours(hour[,min[,sec[,ms]]])`

Sets the `[Value]` property of this value to UTC time. Returns the value of the `[Value]` property of this value. When entering a value for hours, an hour value greater than 23 is added to the existing hour value, not set.

setMilliseconds(ms)

`setMilliseconds(ms)`

Computes UTC from argument and sets the `[Value]` property of this value to `TimeClip(calculatedUTCtime)`. Returns the value of the `[Value]` property of this value.

setMinutes(min[,sec[,ms]])

`setMinutes(min[,sec[,ms]])`

Sets the `[Value]` property of this value to UTC time. Returns the value of the `[Value]` property of this value.

setMonth(mon[,date])

`setMonth(mon[,date])`

Sets the [Value] property of this value to UTC ECMAScript.Date. Returns the value of the [Value] property of this value. If the [Value] property of this exceeds 11, the [Value] property for this is added to the existing month, not set.

setSeconds(sec [, ms])

`setSeconds (sec [, ms])`

Sets the [Value] property of this value to UTC time. Returns the value of the [Value] property of this value.

setTime(time)

`setTime (time)`

Sets the [Value] property of this value to TimeClip(time). Returns the value of the [Value] property of this value. The [Value] property of this is a millisecond value that is converted by the TimeClip(time) method.

setUTCDate(date)

`setUTCDate (date)`

Sets the [Value] property of this value to ECMAScript.Date. Returns the value of the [Value] property of this value. If the [Value] property of this exceeds 30 or 31, the [Value] of this is added to the existing date value, not set.

setUTCFullYear(year[,mon[,date]])

`setUTCFullYear (year [, mon [, date]])`

Sets the [Value] property of this value to ECMAScript.Date. Returns the value of the [Value] property of this value.

setUTCHours(min[,sec[,ms]])

`setUTCHours (min [, sec [, ms]])`

Sets the [Value] property of this value to time. Returns the value of the [Value] property of this value. When entering a value for hours, an hour value greater than 23 is added to the existing hour value, not set.

setUTCMilliseconds(ms)

`setUTCMilliseconds (ms)`

Sets the [Value] property of this value to time and returns the value of the [Value] property of this value.

setUTCMinutes(min[,sec[,ms]])

`setUTCMinutes (min [, sec [, ms]])`

Sets the [Value] property of this value to time. Returns the value of the [Value] property of this value.

setUTCMonth(mon[,date])

`setUTCMonth (mon [, date])`

Sets the [Value] property of this value to ECMAScript.Date. Returns the value of the [Value] property of this value. If the [Value] property of this exceeds 11, the [Value] property for this is added to the existing month, not set.

setUTCSeconds(sec [, ms])

`setUTCSeconds (sec [, ms])`

Sets the [Value] property of this value to time. Returns the value of the [Value] property of this value.

setYear(year)

`setYear (year)`

Sets the [Value] property of this value to UTC ECMAScript.Date. Returns the value of the [Value] property of this value.

toLocaleString()

`toLocaleString()`

Returns a string value. The contents of the string are implementation-dependent, but are intended to represent the Date in a convenient, human-readable form appropriate to the geographic or cultural locale.

toString()

`toString()`

Returns this string value. The contents of the string are implementation-dependent, but are intended to represent the Date in a convenient, human-readable form in the current time zone.

toUTCString()

`toUTCString()`

Returns a string value. The contents of the string are implementation-dependent, but are intended to represent the Date in a convenient, human-readable form in UTC.

UTC()

`UTC()`

This method can accept a number of different arguments. The UTC function differs from the Date constructor in two ways: it returns a time value as a number, rather than creating a Date object, and it interprets the arguments in UTC rather than as local time.

valueOf()

`valueOf()`

Returns a number, which is this time value. The valueOf() function is not generic, so it generates a runtime error if the object is not a Date object.

Function Object

Used to work with the Function Object. This section includes the following topics:

- ♦ [“Function\(p1, p2, . . . , pn, body\)” on page 301](#)

- ♦ “length” on page 301
- ♦ “toString()” on page 301

Function(p1, p2, . . . , pn, body)

Function Constructor. The last argument specifies the body (executable code) of a function; any preceding arguments specify formal parameter.

length

The value of the length property is usually an integer that indicates the “typical” number of arguments expected by the function. However, the language permits the function to be invoked with some other number of arguments. The behavior of a function when invoked on a number of arguments other than the number specified by its length property depends on the function.

toString()

`String toString()`

An implementation-dependent representation of the function is returned. This representation has the syntax of a `FunctionDeclaration`. The use and placement of whitespace, line terminators, and semicolons within the representation string is implementation-dependent.

Global

ECMAScript provides certain “top-level” methods and properties, so-called because they are available from any context: They are not parented by any particular object.

This section includes the following topics:

- ♦ “escape(string)” on page 301
- ♦ “eval(x)” on page 301
- ♦ “Infinity” on page 302
- ♦ “isFinite(number)” on page 302
- ♦ “isNaN(value)” on page 302
- ♦ “NaN” on page 302
- ♦ “parseFloat(string)” on page 302
- ♦ “parseInt(string, radix)” on page 302
- ♦ “unescape(string)” on page 302

escape(string)

`String escape()`

The escape function computes a new, URL-legal version of a string in which certain URL-illegal characters have been replaced by hexadecimal escape sequences.

eval(x)

`eval()`

When the eval function is called with one argument x, the following steps are taken:

1. If x is not a string value, return x.

2. Parse *x* as an ECMAScript Program. If the parse fails, generate a runtime error.
3. Evaluate the program from Step 2.
4. If Result(3) is “normal” completion after value “V”, return the value V.
5. Return undefined.

Infinity

A special primitive value representing positive infinity.

isFinite(number)

`isFinite()`

Applies `Number()` to its argument, then returns `false` if the result is NaN, +*, or **; otherwise, returns `true`.

isNaN(value)

`isNaN()`

Returns `true` if the argument evaluates to NaN (“not a number”); otherwise, returns `false`.

NOTE: Any form of logical comparison of NaN against anything else, including itself, returns `false`. Use `isNaN()` to determine whether a variable (or a return value, etc.) is equal to NaN.

NaN

The primitive value NaN represents the set of IEEE standard “Not-a-Number” values.

parseFloat(string)

`number parseFloat()`

Produces a floating-point number by interpretation of the contents of the string argument. If the string cannot be converted to a number, the special value NaN (see “NaN” on page 302) is returned.

parseInt(string, radix)

`number parseInt()`

Produces an integer value dictated by interpretation of the contents of the string argument, according to the specified radix.

unescape(string)

`String unescape()`

Computes a new version of a string value in which escape sequences that might be introduced by the escape function are replaced with the character they represent.

Math Object

All of the Math object’s properties and methods are static, which means you should prepend “Math” to the property or method name in your code. For example, use “Math.PI,” not simply “PI.”

This section includes the following topics:

- ♦ “E” on page 303

- ♦ “LN10” on page 303
- ♦ “LN2” on page 303
- ♦ “LOG2E” on page 303
- ♦ “LOG10E” on page 304
- ♦ “PI” on page 304
- ♦ “SQRT1.2” on page 304
- ♦ “SQRT2” on page 304
- ♦ “abs(x)” on page 304
- ♦ “acos(x)” on page 304
- ♦ “asin(x)” on page 304
- ♦ “atan(x)” on page 304
- ♦ “atan2(x,y)” on page 305
- ♦ “ceil(x)” on page 305
- ♦ “cos(x)” on page 305
- ♦ “exp(x)” on page 305
- ♦ “floor(x)” on page 305
- ♦ “log(x)” on page 306
- ♦ “max(x,y)” on page 306
- ♦ “min(x,y)” on page 306
- ♦ “pow(x,y)” on page 306
- ♦ “random()” on page 306
- ♦ “round(x)” on page 306
- ♦ “sin(x)” on page 307
- ♦ “sqrt(x)” on page 307
- ♦ “tan(x)” on page 307

E

The number value for e, the base of the natural logarithms, which is approximately 2.7182818284590452354.

LN10

The number value for the natural logarithm of 10, which is approximately 2.302585092994046.

LN2

The number value for the natural logarithm of 2, which is approximately 0.6931471805599453.

LOG2E

The number value for the base-2 logarithm of e, the base of the natural logarithms; this value is approximately 1.4426950408889634. The value of Math.LOG2E is approximately the reciprocal of the value of Math.LN2.

LOG10E

The number value for the base-10 logarithm of e, the base of the natural logarithms; this value is approximately 0.4342944819032518. The value of Math.LOG10E is approximately the reciprocal of the value of Math.LN10.

PI

The number value for π , the ratio of the circumference of a circle to its diameter, which is approximately 3.14159265358979323846.

SQRT1_2

The number value for the square root of 1/2, which is approximately 0.7071067811865476. The value of Math.SQRT1_2 is approximately the reciprocal of the value of Math.SQRT2.

SQRT2

The number value for the square root of 2, which is approximately 1.4142135623730951.

abs(x)

Number abs(x)

Returns the absolute value of the argument x; in general, the result has the same magnitude as the argument but has positive sign. The input value x can be any number value.

Example:

```
Math.abs(-123.23940) = 123.23940
```

acos(x)

Number acos(x)

This function returns an implementation-dependent approximation to the arc cosine of the argument. The result is expressed in radians and ranges from +0 to +PI(3.14159...) radians. The input value x must be a number between -1.0 and 1.0.

Example:

```
PI/4 = 0.785 Math.acos(0.785) = 0.6681001997570769
```

asin(x)

Number asin(x)

This function returns an implementation-dependent approximation to the arc sine of the argument. The result is expressed in radians and ranges from -PI/2 to +PI/2. The input value x must be a number between -1.0 and 1.0.

Example:

```
PI/4 = 0.785 Math.asin(0.785) = 0.9026961270378197
```

atan(x)

Number atan(x)

This function returns an implementation-dependent approximation to the arc tangent of the argument. The result is expressed in radians and ranges from -PI/2 to +PI/2. The input value x can be any number.

Example:

```
3PI/4 = 2.355 Math.atan(2.355) = 1.169240427545485
```

atan2(x,y)

Number atan2(x,y)

This function returns an implementation-dependent approximation to the arc tangent of the quotient y/x of the arguments y and x , where the signs of the arguments are used to determine the quadrant of the result. It is intentional and traditional for the two-argument arc tangent function that the argument named y be first and the argument named x be second. The result is expressed in radians and ranges from $-\pi$ to $+\pi$. The input value x is the x -coordinate of the point. The input value y is the y -coordinate of the point.

Example:

```
PI/2 = 1.57 Math.atan2(1.57,-1.57) = 2.356194490192345
```

ceil(x)

Number ceil(x)

This function returns the smallest (closest to $-\infty$) number value that is not less than the argument and is equal to a mathematical integer. If the argument is already an integer, the result is the argument itself. The input value x can be any numeric value or expression. The `Math.ceil(x)` function property is the same as `-Math.floor(-x)`. Example:

Example:

```
Math.ceil(123.78457) = 123
```

cos(x)

Number cos(x)

This function returns an implementation-dependent approximation to the cosine of the argument. The argument must be expressed in radians.

exp(x)

Number exp(x)

This function returns an implementation-dependent approximation to the exponential function of the argument (e raised to the power of the argument, where e is the base of the natural logarithms). The input value x can be any numeric value or expression greater than 0.

Example:

```
Math.exp(10) = 22026.465794806718
```

floor(x)

Number floor(x)

This function returns the greatest (closest to $+\infty$) number value that is not greater than the argument and is equal to a mathematical integer. If the argument is already an integer, the result is the argument itself. The input value x can be any numeric value or expression.

Example:

```
Math.floor(654.895869)=654
```

log(x)

Number log(x)

This function returns an implementation-dependent approximation to the natural logarithm of the argument. The input value x can be any numeric value or expression greater than 0.

Example:

```
Math.log(2) = 0.6931471805599453
```

max(x,y)

Number max(x, y)

This function returns the larger of the two arguments. The input values x and y can be any numeric values or expressions.

Example:

```
Math.max(12.345, 12.3456) = 12.3456
```

min(x,y)

Number min(x, y)

This function returns the smaller of the two arguments. The input values x and y can be any numeric values or expressions.

Example:

```
Math.min(-12.457, -12.567) = -12.567
```

pow(x,y)

Number pow(x, y)

This function returns an implementation-dependent approximation to the result of raising x to the power of y. The input value x must be the number raised to a power. The input value y must be the power to which x is raised.

Example:

```
Math.pow(2, 4) = 16
```

random()

Number random()

This method takes no arguments and returns a pseudo-random number between 0 and 1. The number value has approximately uniform distribution over that range, using an implementation-dependent algorithm or strategy. This function takes no arguments.

Example:

```
Math.random() = 0.9545176397178535
```

round(x)

Number round(x)

This function returns the number value that is closest to the argument and is equal to a mathematical integer. If two integer number values are equally close to the argument, then the result is the number value that is closer to +infinity. If the argument is already an integer, the result is the argument itself. The input value x can be any number.

Example:

```
Math.round(13.53) = 14
```

sin(x)

Number sin(x)

This function returns an implementation-dependent approximation to the sine of the argument. The argument is expressed in radians. The input value x must be an angle measured in radians.

sqrt(x)

Number sqrt(x)

This function returns an implementation-dependent approximation to the square root of the argument. The input value x must be any numeric value or expression greater than or equal to 0. If the input value x is less than zero, the string “NaN” is returned. (NaN stands for “Not a Number”.)

Example:

```
Math.sqrt(25) = 5
```

tan(x)

Number tan(x)

This function returns an implementation-dependent approximation to the tangent of the argument. The argument is expressed in radians. The input value x must be an angle measured in radians.

Number Object

Lets you work with numeric values. The Number object is an object wrapper for primitive numeric values.

This section includes the following topics:

- ♦ “MAX_VALUE” on page 307
- ♦ “MIN_VALUE” on page 307
- ♦ “NaN” on page 308
- ♦ “NEGATIVE_INFINITY” on page 308
- ♦ “Number()” on page 308
- ♦ “POSITIVE_INFINITY” on page 308
- ♦ “toString(radix)” on page 308
- ♦ “valueOf()” on page 308

MAX_VALUE

The largest positive finite value of the number type (approximately 1.7976931348623157e308).

Example:

```
Number.MAX_VALUE
```

MIN_VALUE

The smallest positive nonzero value of the number type (approximately 5e-324).

Example:

```
Number.MIN_VALUE
```

NaN

The primitive value NaN represents the set of IEEE Standard “Not-a-Number” values.

Example:

```
Number.NaN
```

NEGATIVE_INFINITY

The value of negative infinity.

Example:

```
Number.NEGATIVE_INFINITY
```

Number()

```
Number()
```

The constructor of Number has two forms: Number(value) and Number().

POSITIVE_INFINITY

The value of positive infinity.

Example:

```
Number.POSITIVE_INFINITY
```

toString(radix)

```
toString()
```

If the radix is the number 10 or is not supplied, then this number value is given as an argument to the ToString operator; the resulting string value is returned. If the radix is supplied and is an integer from 2 to 36, but not 10, the result is a string, the choice of which is implementation-dependent. The toString function is not generic; it generates a runtime error if this value is not a Number object. Therefore, it cannot be transferred to other kinds of objects for use as a method.

valueOf()

```
valueOf()
```

Returns this number value. The valueOf function is not generic; it generates a runtime error if its value is not a Number object. Therefore, it cannot be transferred to other kinds of objects for use as a method.

Object

Used to work with objects. Object is the primitive JavaScript object type. All ECMAScript objects are descended from object. That is, all ECMAScript objects have the methods defined for object.

This section includes the following topics:

- ♦ “Object()” on page 309
- ♦ “toString()” on page 309

- ♦ “valueOf()” on page 309

Object()

Constructor for object.

toString()

`Object toString()`

When the toString method is called on an arbitrary object, the following steps are taken:

1. Get the `[[Class]]` property of this object.
2. Compute a string value by concatenating the three strings “[object “, `Result(1)`, and “]”.
3. Return `Result(2)`.

valueOf()

`Object valueOf()`

The valueOf method for an object usually returns the object; however, if the object is a “wrapper”

for a host object, as might be created by the Object constructor, the contained host object should be returned.

String Object

Used to work with String Objects. This section includes the following topics:

- ♦ “String(x)” on page 309
- ♦ “charAt(pos)” on page 310
- ♦ “charCodeAt(pos)” on page 310
- ♦ “fromCharCode(char0, char1, . . .)” on page 310
- ♦ “indexOf(searchString, pos)” on page 310
- ♦ “lastIndexOf(searchString, pos)” on page 310
- ♦ “length” on page 310
- ♦ “match(RegExp)” on page 310
- ♦ “replace(RegExp, String)” on page 310
- ♦ “search(RegExp)” on page 311
- ♦ “split(separator)” on page 311
- ♦ “substring(start, end)” on page 311
- ♦ “toLowerCase()” on page 311
- ♦ “toString()” on page 311
- ♦ “toUpperCase()” on page 311
- ♦ “valueOf()” on page 311

String(x)

`String(x)`

The constructor of the string.

charAt(pos)

`charAt(pos)`

Returns a string containing the character at position `pos` in the string resulting from converting this object to a string. If there is no character at that position, the result is the empty string. The result is a string value, not a string object.

charCodeAt(pos)

`charCodeAt(pos)`

Returns a number (a nonnegative integer less than 2^{16}) representing the Unicode* code point encoding of the character at position `pos` in the string resulting from converting this object to a string. If there is no character at that position, the result is NaN.

fromCharCode(char0, char1, ...)

`fromCharCode(char0, char1, . . .)`

Returns a string value containing as many characters as the number of arguments. Each argument specifies one character of the resulting string, with the first argument specifying the first character, and so on, from left to right. An argument is converted to a character by applying the operation `ToUint16` and regarding the resulting 16-bit integer as the Unicode code point encoding of a character. If no arguments are supplied, the result is the empty string.

indexOf(searchString, pos)

`indexOf(searchString, pos)`

If the given `searchString` appears as a substring of the result of converting this object to a string, at one or more positions that are at or to the right of the specified position, then the index of the leftmost such position is returned; otherwise, -1 is returned. If position is undefined or not supplied, 0 is assumed, in order to search all of the string.

lastIndexOf(searchString, pos)

`lastIndexOf(searchString, pos)`

If the given `searchString` appears as a substring of the result of converting this object to a string, at one or more positions that are at or to the left of the specified position, then the index of the rightmost such position is returned; otherwise, -1 is returned. If position is undefined or not supplied, the length of the string value is assumed, in order to search all of the string.

length

Returns the length of the String.

match(RegExp)

`String match(RegExp)`

Takes a regular expression object as argument. It returns an array of matches; otherwise, returns null.

replace(RegExp, String)

`String replace(RegExp, String)`

Takes a regular expression and a replacement string. It returns the original string with replacements accomplished.

search(RegExp)

`String search(RegExp)`

Takes a regular expression as the sole arg and returns the offset of the first substring that matches, or -1 on no match.

split(separator)

`split(separator)`

Returns an Array object, into which substrings of the result of converting this object to a string have been stored. The substrings are determined by searching from left to right for occurrences of the given separator; these occurrences are not part of any substring in the returned array, but serve to divide the string value. The separator may be a string of any length.

substring(start, end)

`substring(start, end)`

Returns a substring of the result of converting this object to a string, starting from character position start and running to the position end of the string. If the second parameter is not present, the end position is considered the end of the string. The result is a string value, not a string object.

toLowerCase()

`toLowerCase()`

Returns a string equal in length to the length of the result of converting this object to a string. The result is a string value, not a string object. Every character of the result is equal to the corresponding character of the string, unless that character has a Unicode 2.0 lowercase equivalent, in which case the lowercase equivalent is used instead. The canonical Unicode 2.0 case mapping must be used, which does not depend on implementation or locale.

toString()

`toString()`

Returns this string value. When concerned with the placement and use of whitespace line terminators and semicolons within the representation, the string value is implementation-dependent.

toUpperCase()

`toUpperCase()`

Returns a string equal in length to the length of the result of converting this object to a string. The result is a string value, not a string object. Every character of the result is equal to the corresponding character of the string, unless that character has a Unicode 2.0 uppercase equivalent, in which case the uppercase equivalent is used instead. The canonical Unicode 2.0 case mapping must be used, which does not depend on implementation or locale.

valueOf()

`valueOf()`

Returns this string value. The valueOf() function is not generic, so it generates a runtime error if the object is not a String object.

10.3.4 Global Functions

Global functions in ECMAScript are functions that are independent of any particular object.

The Expression Builder supports the following Global functions:

getEnvironmentCountry

```
getEnvironmentCountry()
```

Function returns a 2-character string (for example, US) that represents the location that is currently selected on the user's computer.

getEnvironmentLanguage

```
getEnvironmentLanguage()
```

Function returns a 2-character string (for example, EN) that represents the input language that is currently selected on the user's computer.

10.3.5 IDVault Functions

This section lists functions that are used with IDVault data.

DNCompare

```
DNcompare(String dn1, String dn2)
```

Performs a case-insensitive comparison of DNs from the Identity Vault. Returns true if the DNs are the same.

A DN encapsulates a Distinguished Name (an LDAP name with context). The syntax of the DNs must conform to that specified in RFC 2253, which describes the String representation of DNs. The following DNs are all valid for use with DNCompare (and would return true if compared):

```
cn=jdoe,ou=users,ou=idmsample,o=acme  
CN=jdoe,ou=users,ou=idmsample,o=acme  
cn=JDOE,ou=users,ou=idmsample,o=acme
```

For more information about RFC 2253, see [RFC 2353 \(http://www.ietf.org/rfc/rfc2253.txt\)](http://www.ietf.org/rfc/rfc2253.txt).

Example:

```
if ( IDVault.DNcompare(flowdata.get('Activity3/CardRequest/  
Candidate'),recipient )) true; else false ;
```