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 © 2007 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 About the Provisioning Team Editor .......................... 14
  1.6 Documenting a Project ..................................... 15

2 Working with the Provisioning View 17
  2.1 Setting Up a Provisioning Project ................................ 17
  2.2 Accessing the Provisioning View ................................ 22
  2.3 Setting Provisioning View Preferences .......................... 23
  2.4 Importing Provisioning Objects ................................ 26
  2.5 Exporting Provisioning Objects ................................ 27
  2.6 Validating Provisioning Objects ................................ 27
  2.7 Deploying Provisioning Objects ................................ 28
  2.8 Comparing Provisioning Objects ................................ 30
  2.9 Specifying Locales and Localization Resource Groups ......... 31
  2.10 Localizing Display Labels .................................... 36

3 Configuring the Directory Abstraction Layer 41
  3.1 About the Directory Abstraction Layer .......................... 41
  3.2 Working with Entities and Attributes ........................... 45
  3.3 Working with Lists ............................................ 53
  3.4 Working with Queries ........................................... 57
  3.5 Working with Relationships ....................................... 60
  3.6 Working with Configuration Settings ............................ 63
## 4 Working with the Provisioning Request Definition Editor

4.1 About the Provisioning Request Definition Editor .............................................. 81
   4.1.1 How the Provisioning Request Definition Editor Fits into the Identity Manager Architecture .................................................. 81
   4.1.2 Provisioning and Workflow Example ................................................................ 82
4.2 Basic Steps for Creating a Provisioning Request Definition ................................ 87
4.3 Guidelines for Creating Workflows ................................................................. 88
   4.3.1 Rules for Activities ................................................................................... 88
   4.3.2 Rules for Flow Paths ............................................................................... 89
   4.3.3 Understanding Workflow Data ................................................................ 90
4.4 Working with the Installed Templates .......................................................... 94
4.5 Debugging a Workflow .................................................................................. 96

## 5 Creating a Provisioning Request Definition

5.1 About the Wizard and the Overview Tab .................................................... 99
5.2 Using the Wizard to Create a Provisioning Request Definition ................. 101
   5.2.1 Using a Template .................................................................................. 102
   5.2.2 From Concept to Finished Product ...................................................... 104
   5.3 Using the Overview Tab to Modify Basic Settings .................................... 105

## 6 Creating Forms for a Provisioning Request Definition

6.1 About Forms .................................................................................................. 109
   6.1.1 About Form Control Data Binding ......................................................... 111
   6.1.2 About Forms and Events ..................................................................... 111
6.2 About the Forms Tab .................................................................................... 112
   6.2.1 About Form Selection .......................................................................... 113
   6.2.2 About Form Controls .......................................................................... 114
6.3 Creating Forms .............................................................................................. 116
   6.3.1 Creating New Forms ........................................................................... 116
   6.3.2 Adding Form Controls and Actions .................................................... 117
   6.3.3 Defining Events ................................................................................... 119
   6.3.4 Using the Scripts Tab .......................................................................... 123
6.4 Action Reference ............................................................................................ 125
6.5 Form Control Reference ............................................................................... 127
   6.5.1 Controls for User-Entered Comments .................................................. 129
   6.5.2 General Form Control Properties ....................................................... 129
   6.5.3 CheckBoxPickList ............................................................................... 130
   6.5.4 DatePicker ......................................................................................... 131
   6.5.5 DateTimePicker ..................................................................................... 132
   6.5.6 DNContainer ....................................................................................... 133
   6.5.7 DNDisplay ......................................................................................... 134
   6.5.8 DNLookup ......................................................................................... 135
   6.5.9 DNMaker ............................................................................................ 139
   6.5.10 DNQuery ............................................................................................ 140
   6.5.11 Global List .......................................................................................... 141
   6.5.12 Html .................................................................................................... 142
   6.5.13 MVCheckbox ....................................................................................... 142
7 Creating the Workflow for a Provisioning Request Definition

7.1 About the Workflow Tab ........................................ 165
  7.1.1 Canvas .................................................. 166
  7.1.2 Palette ............................................... 167
  7.1.3 Views .................................................. 169
7.2 Adding Activities to a Workflow ........................................ 170
  7.2.1 Setting the General Properties of an Activity ............... 170
  7.2.2 Defining the Data Item Mappings .......................... 171
  7.2.3 Defining the E-Mail Notification Settings .................. 172
7.3 Adding Flow Paths ............................................... 173
7.4 Configuring Flow Paths ........................................... 174
7.5 Addressing an Approval Activity .................................. 176
  7.5.1 Valid Addressee Expressions .................................. 177
  7.5.2 Relationship Between Addressee and Approver Type ......... 178
7.6 Provisioning Multiple Individuals with One Workflow Instance ........................................ 181
  7.6.1 Basic Steps for Using the Workflow ......................... 181
  7.6.2 Setting up the Workflow for a Team Manager to Use ....... 183
7.7 Working with Entity Activities ..................................... 183
  7.7.1 Adding or Modifying an Entity ............................... 183
  7.7.2 Using an Entity Activity to Delete an Entity .................. 184
  7.7.3 Using an Entity Activity to Delete an Attribute or Value .. 184
7.8 Making Distinguished Name References Portable .................... 185
7.9 Configuring Digital Signature Support ........................ 185
  7.9.1 Digital Signature Workflow Properties ................. 186
  7.9.2 Creating a Signature Declaration ....................... 187

8 Workflow Activity Reference ........................................ 189

8.1 Start Activity .................................................. 189
  8.1.1 Properties ............................................. 190
  8.1.2 Data Item Mapping ...................................... 191
  8.1.3 E-Mail Notification ...................................... 191
8.2 Approval Activity ............................................... 191
  8.2.1 Properties ............................................. 192
  8.2.2 Data Item Mapping ...................................... 198
  8.2.3 E-Mail Notification ...................................... 198
8.3 Log Activity .................................................. 200
  8.3.1 Properties ............................................. 201
  8.3.2 Data Item Mapping ...................................... 201
  8.3.3 E-Mail Notification ...................................... 201
8.4 Branch Activity ................................................................. 202
  8.4.1 Properties ........................................................................ 202
  8.4.2 Data Item Mapping .......................................................... 202
  8.4.3 E-Mail Notification .......................................................... 202
8.5 Merge Activity ................................................................. 202
  8.5.1 Properties ........................................................................ 203
  8.5.2 Data Item Mapping .......................................................... 203
  8.5.3 E-Mail Notification .......................................................... 203
8.6 Condition Activity ......................................................... 203
  8.6.1 Properties ........................................................................ 203
  8.6.2 Data Item Mapping .......................................................... 204
  8.6.3 E-Mail Notification .......................................................... 204
8.7 Mapping Activity ............................................................. 204
  8.7.1 Properties ........................................................................ 204
  8.7.2 Data Item Mapping .......................................................... 205
  8.7.3 E-mail Notification .......................................................... 205
8.8 Workflow Status ............................................................... 205
  8.8.1 Properties ........................................................................ 205
  8.8.2 Data Item Mapping .......................................................... 206
  8.8.3 E-Mail Notification .......................................................... 206
8.9 E-Mail Activity ................................................................. 206
  8.9.1 Properties ........................................................................ 206
  8.9.2 Data Item Mapping .......................................................... 207
  8.9.3 E-Mail Notification .......................................................... 207
8.10 Finish Activity ................................................................. 208
  8.10.1 Properties ........................................................................ 209
  8.10.2 Data Item Mapping .......................................................... 209
  8.10.3 E-mail Notification .......................................................... 209
8.11 Integration Activity ......................................................... 210
  8.11.1 Properties ........................................................................ 211
  8.11.2 Data Item Mapping .......................................................... 211
  8.11.3 E-Mail Notification .......................................................... 212
8.12 Entitlement Activity ......................................................... 213
  8.12.1 Properties ........................................................................ 213
  8.12.2 Data Item Mapping .......................................................... 213
  8.12.3 E-Mail Notification .......................................................... 214
8.13 Entity Activity ................................................................. 214
  8.13.1 Properties ........................................................................ 214
  8.13.2 Data Item Mapping .......................................................... 215
  8.13.3 E-Mail Notification .......................................................... 217

9 Working with Integration Activities ........................................................................ 219
  9.1 About the Integration Activity ............................................. 219
  9.2 Adding an Integration Activity ............................................. 219
  9.3 Moving Data to and from the Integration Activity ............... 221
  9.4 Using the Integration Activity Editor Interface ................. 224
    9.4.1 XML Views .................................................................... 225
    9.4.2 Action Model ................................................................. 229
    9.4.3 WSDL Editor ................................................................. 236
    9.4.4 Messages ...................................................................... 236
    9.4.5 Regenerating Code for the Action Model ...................... 236
    9.4.6 Adding Actions to the Action Model ......................... 236
  9.5 Actions ............................................................................... 237
    9.5.1 Advanced ...................................................................... 237
    9.5.2 Data Exchange .............................................................. 243
9.5.3 Repeat ................................................................. 248
9.5.4 Comment ........................................................... 255
9.5.5 Decision .............................................................. 256
9.5.6 Function .............................................................. 257
9.5.7 Log ................................................................. 258
9.5.8 Map ................................................................. 259

10 Working with ECMA Expressions ........................................ 267
10.1 About the ECMA Expression Builder .................................... 267
10.1.1 About ECMAScript .................................................. 268
10.1.2 ECMAScript Capabilities ........................................... 268
10.1.3 Using the ECMA Expression Builder ................................ 268
10.1.4 About Java Integration ............................................ 274
10.1.5 About XPath Integration ........................................... 275
10.1.6 Performance Considerations ....................................... 276
10.2 ECMAScript Examples .................................................. 277
10.2.1 General Examples .................................................. 277
10.2.2 Flowdata Examples .................................................. 278
10.2.3 Form Control Examples ............................................ 278
10.2.4 Error Handling ..................................................... 279
10.3 ECMAScript API ....................................................... 280
10.3.1 Form Action Script Methods ....................................... 280
10.3.2 DOM Methods ..................................................... 290
10.3.3 ECMAScript Core .................................................. 314
10.3.4 Global Functions ................................................... 333
10.3.5 IDVault Functions .................................................. 333

11 Configuring Provisioning Teams .......................................... 335
11.1 About Teams ......................................................... 335
11.1.1 About Provisioning Team Requests ................................ 336
11.1.2 Using a Team to Manage Direct Reports ............................ 336
11.2 Managing Provisioning Teams .......................................... 337
11.2.1 Creating a Provisioning Team ...................................... 337
11.2.2 Deleting a Provisioning Team ...................................... 343
11.2.3 Creating a Team to Manage Direct Reports ......................... 343
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.

- Chapter 1, “Introduction to the User Application Design Tools,” on page 13
- Chapter 2, “Working with the Provisioning View,” on page 17
- Chapter 3, “Configuring the Directory Abstraction Layer,” on page 41
- Chapter 4, “Working with the Provisioning Request Definition Editor,” on page 81
- Chapter 5, “Creating a Provisioning Request Definition,” on page 99
- Chapter 6, “Creating Forms for a Provisioning Request Definition,” on page 109
- Chapter 7, “Creating the Workflow for a Provisioning Request Definition,” on page 165
- Chapter 8, “Workflow Activity Reference,” on page 189
- Chapter 9, “Working with Integration Activities,” on page 219
- Chapter 10, “Working with ECMA Expressions,” on page 267
- Chapter 11, “Configuring Provisioning Teams,” on page 335

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).

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

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, “About the Provisioning Team Editor,” on page 14
- Section 1.6, “Documenting a Project,” on page 15

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:

- Access the editors that allow you to create and manipulate User Application components, such as:
  - The directory abstraction layer editor.
  - The provisioning request definitions editor.
  - The provisioning team editor.

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

- Manipulate object definitions, such as:
  - Importing and exporting object definitions from the Identity Vault or the local file system.
  - Validating local object definitions.
  - Deploying object definitions to the Identity Vault.
  - Comparing the objects on the local file system with those in the Identity Vault.

- Define the User Application driver’s supported and default locales including:
  - Importing and exporting display labels and other User Application strings for localization.
  - Defining custom localization resource groups (used only for field localization).

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 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 workflow path by using the Condition Activity.
  - Write different log messages that are conditionally triggered by using a single Log Activity.

1.5 About the Provisioning Team Editor

The provisioning team editor allows you to define a set of users who can act as a team for the Requests & Approvals tab of the User Application. The team definition determines who can manage provisioning requests and approval tasks associated with this team. The team definition consists of a list of team managers, team members, and team options. In addition, you can define the set of provisioning request objects that the team can act on.
1.6 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. The sections include:

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 provisioning request definition documentation 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.

The provisioning team documentation includes:

- Display name
- The team members
- The request type and scope
- The manager’s permissions
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 23
- Section 2.4, “Importing Provisioning Objects,” on page 26
- Section 2.5, “Exporting Provisioning Objects,” on page 27
- Section 2.6, “Validating Provisioning Objects,” on page 27
- Section 2.7, “Deploying Provisioning Objects,” on page 28
- Section 2.8, “Comparing Provisioning Objects,” on page 30
- Section 2.9, “Specifying Locales and Localization Resource Groups,” on page 31
- Section 2.10, “Localizing Display Labels,” on page 36

### 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 3.5: 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.

![Project view](image)

2. Double-click *System Model*.
3 Access the driver configuration page for a new driver by 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.

4 Select one of these driver configurations.

<table>
<thead>
<tr>
<th>Driver Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Application 3.0.1</td>
<td>Creates a Version 3.0.1 User Application driver.</td>
</tr>
<tr>
<td>User Application 3.5</td>
<td>Creates a Version 3.5 User Application driver.</td>
</tr>
<tr>
<td>User Application 3.5.1</td>
<td>Creates a Version 3.5.1 User Application driver.</td>
</tr>
</tbody>
</table>

This procedure describes how to configure all versions and indicates where fields are specific to one version or another.
Designer launches the following:

![Image of Import Information Requested dialog box]

5 Fill in the fields as follows:

<table>
<thead>
<tr>
<th>Property</th>
<th>What to Specify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver Name</td>
<td>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.</td>
</tr>
<tr>
<td>Authentication ID</td>
<td>The DN of the User Application Administrator.</td>
</tr>
</tbody>
</table>
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.
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**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Indicates that the local object has changed.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Indicates that the local object contains a validation warning.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Indicates that the local object contains a validation error.</td>
</tr>
</tbody>
</table>

**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. Figure 2-2 shows an example of the tooltip for a Version 3.5 User Application driver.

**Figure 2-2  User Application Driver Tooltip**

![Tooltip](image)

**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>
<thead>
<tr>
<th>Preference Category</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Prompt for deletion of User Application Configuration</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Set delete from Identity Vault as default for all “Confirm Delete” dialogs</td>
<td>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 Delete object in Identity Vault on deploy 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.</td>
</tr>
<tr>
<td></td>
<td>Show Provisioning View when new User Application is created or imported</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Validate display names for supported locales</td>
<td>Select this option if you want Designer to validate display names to ensure:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>† Uniqueness of a display name within a locale.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>† That a display name is supplied (not blank) for each locale.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Applies to display names defined by using the directory abstraction layer editor, provisioning request editor, or provisioning teams editor.</td>
</tr>
<tr>
<td></td>
<td>Show categories in Provisioning View</td>
<td>Select this option so Designer displays provisioning request definitions organized by category. You specify the category in the Overview panel. Categories are defined in the Provisioning Category list defined in the directory abstraction layer.</td>
</tr>
<tr>
<td></td>
<td>Show all localized e-mail templates</td>
<td>Select this option so Designer displays all localized e-mail templates as selectable options in the E-Mail notification tab. The Java language code is appended to the name of the e-mail template. For example, cn=Provisioning Notification Activity_es, cn=Default Notification Collection,cn=security indicates this is the Spanish language version of this template. When you select a localized template, that language is used regardless of the user’s default language. When you select the default template (the template without a locale code), the e-mail is in the user’s default language (if the default is a supported language).</td>
</tr>
<tr>
<td></td>
<td>Identity Vault Connection Timeout (in milliseconds)</td>
<td>The amount of time (in milliseconds) for Designer to connect to the Identity Vault. If it 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 through the ECMA expression builder.</td>
</tr>
</tbody>
</table>
## Working with the Provisioning View

### Preference Category: Import/Deploy

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Import</strong></td>
<td><strong>Delete local object on import when object has been deleted in Identity Vault</strong>: Select this option for 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.</td>
</tr>
<tr>
<td><strong>Prompt whether to overwrite runtime configuration on import from file</strong>: 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.</td>
<td></td>
</tr>
<tr>
<td><strong>Deploy</strong></td>
<td><strong>Allow deployment of objects with validation errors</strong>: 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, &quot;Validating Provisioning Objects,&quot; on page 27. Deselect this option to prevent deployment of definitions that fail validation.</td>
</tr>
<tr>
<td></td>
<td><strong>WARNING</strong>: Deploying objects that fail validation can result in errors in the User Application runtime.</td>
</tr>
</tbody>
</table>

### Preference Category: Migration

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show warning about Identity Vault schema changes</strong></td>
<td>When you select the Migrate command, Designer displays a dialog box that warns you that schema changes (that are needed to support new features) must be 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 Migrate command, deselect this option.</td>
</tr>
<tr>
<td><strong>Always deploy (undeleted) User Application Driver</strong></td>
<td>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 <strong>Always deploy (undeleted) User Application driver</strong> option if you always want Designer to deploy the User Application driver, and you don’t want this dialog box displayed.</td>
</tr>
<tr>
<td><strong>Show warning that editors will be closed</strong></td>
<td>When you select the Migrate 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 Migrate command.</td>
</tr>
</tbody>
</table>
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*.

---

<table>
<thead>
<tr>
<th>Preference Category</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation Mask</td>
<td>Validation Mask Table</td>
<td>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. 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 <em>Restore Defaults</em>, then clicking <em>Apply</em>.</td>
</tr>
<tr>
<td>Workflows</td>
<td>Form Templates</td>
<td>Use this dialog box to remove or preview existing form templates.</td>
</tr>
</tbody>
</table>
|                     | Diagram Preferences | **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.  
                  **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. |

---

**Validation Mask**

**Mask**

**Validation Mask Table**

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.

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 *Restore Defaults*, then clicking *Apply*.

---

**Workflows**

**Form Templates**

Use this dialog box to remove or preview existing form templates.

---

**Diagram Preferences**

**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.

**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.
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*.

   **NOTE:** For provisioning teams, *Import Object* imports only the team object. *Import Team Requests* imports any associated team request objects.

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:

- 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:
• 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=died, 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.

For Provisioning Team objects, Designer:
• Validates that managers and members have been defined for the team.
• Validates that team requests are specified for the team.
• If the request scope is Categories, it validates that the team request actually references a
  category.

For the User Application driver locales, Designer ensures that the locales contain descriptions and
  display names. You can turn off the validation of display names for each locale by setting a
  preference. For more information, see Section 2.3, “Setting Provisioning View Preferences,” on
  page 23.

To validate objects from the:
• Provisioning view, right-click a node and click Validate.
• Directory abstraction layer editor, click Validate Abstraction Layer from the editor’s toolbar, or
  select DAL > Validate from Designer’s menu.
• Provisioning request definition editor, select PRD > Validate from Designer’s menu.
• Provisioning team editor, select Team Editor > 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 the items 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 and then 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** that contains provisioning objects that fail validation, Designer deploys the driver but not the invalid objects (regardless of the deployment preferences). Designer displays the errors 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.

**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, 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. If this is the case, correct the error, then deploy the provisioning request definition again.
• Designer cannot evaluate expressions at design time, so it might display a warning when you use 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. You can review 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:


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.

NOTE: For provisioning teams, you must select the container to compare the provisioning request and provisioning team objects. If you select an individual team compares only the provisioning team objects.

3. If there are differences, select one of the following actions:

<table>
<thead>
<tr>
<th>Reconcile Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do not reconcile</td>
<td>Do not change any definitions.</td>
</tr>
<tr>
<td>Update Designer</td>
<td>Import the definitions from the Identity Vault.</td>
</tr>
</tbody>
</table>
2.9 Specifying Locales and Localization Resource Groups

You can use Designer to define:

- The User Application driver’s default locale. It is the locale used to display content when a match for the user’s locale cannot be found.
- The locales supported by the User Application driver.
- Custom localization resource groups for specialized field localization.

**NOTE:** The ability to specify locales and localization resource groups is only available for User Application drivers for Version 3.5.1 (and higher).

To specify the User Application driver’s default locale:

1. Right-click the User Application driver in the Provisioning View, then select **Configure > Default Locale**.
2. Select the locale from the drop-down list box, then click **OK**. If you do not see the locale in the list, you must add it through the Locales dialog box.

To define the User Application driver’s supported locales:

---

<table>
<thead>
<tr>
<th>Reconcile Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update eDirectory</td>
<td>Deploy the definition from Designer to the Identity Vault.</td>
</tr>
<tr>
<td>Reconciled by parent</td>
<td>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.</td>
</tr>
</tbody>
</table>
1 Right-click the User Application driver in the Provisioning view, then select Configure > Locales. Designer displays the Supported Locales dialog box.

![Supported Locales dialog box]

2 Fill in the fields as follows:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="add_icon" alt="Add" /></td>
<td>Lets you add a new locale to the list of locales supported by the User Application driver. When you click the Add button, Designer prompts you for the Language and Country. You can either select them or type a value. The Language is required and displays as the locale. The country is not required.</td>
</tr>
<tr>
<td><img src="remove_icon" alt="Remove" /></td>
<td>Deletes the selected Locale from the supported list. Any files for the selected locale are not deleted.</td>
</tr>
</tbody>
</table>
Click OK.

To create a custom localization resource group:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Localize all supported locales" /></td>
<td>Lets you localize all supported locales in one dialog box. When you click the button, it launches this dialog box:</td>
</tr>
<tr>
<td><img src="image" alt="Select source and target languages" /></td>
<td>You can select the source language from which you want to translate the names of the supported locales. Select the target language into which you want to translate the names of the supported locales. Type the translation in the Target field.</td>
</tr>
<tr>
<td><img src="image" alt="Localization" /></td>
<td>Lets you localize the display labels for the selected locale. This is the same procedure described in Section 2.10, “Localizing Display Labels,” on page 36.</td>
</tr>
</tbody>
</table>

3 Click OK.
1 Right-click the User Application driver in the Provisioning view, then select Configure > Locales. Go to the Resource Groups tab.

2 Click ![image](image.png)

3 Complete the fields as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>A unique name used to identify the localization resource group.</td>
</tr>
<tr>
<td>Display Name</td>
<td>The name displayed in Designer to correspond with this group.</td>
</tr>
</tbody>
</table>
4 Click OK. The Resource Groups property dialog box displays:

5 Complete the fields as follows:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Add Button" /></td>
<td>Lets you add a new localization resource group.</td>
</tr>
<tr>
<td><img src="image" alt="Delete Button" /></td>
<td>Deletes the current localization resource group.</td>
</tr>
<tr>
<td><img src="image" alt="Edit Button" /></td>
<td>Lets you localize the localization resource groups in one dialog box.</td>
</tr>
</tbody>
</table>

**Description**

Lets you add descriptive text for this entry.

**Required Group**

Lets you specify dependencies on other localization resource groups.

Localization resource groups are used for creating resource bundles for non-standard language. The required groups are defined in the *Identity Manager 3.5.1 User Application: Localization Toolkit Guide*. Contact your Novell Sales Representative for more information about the toolkit.

**Locales**

Lets you add or remove the locales into which this localization resource group must be localized. If the locale is not on the list, see “To define the User Application driver’s supported locales:” on page 31.
2.10 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*

![Localization Dialog Box](image)

<table>
<thead>
<tr>
<th>Table 2-3  Localizable Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Designer tool</strong></td>
</tr>
</tbody>
</table>
| Directory Abstraction Layer Editor | • Entity and attribute display labels  
• Relationship names  
• Global and local list items  
• Query display labels and parameter display labels. |
| Provisioning Request Definition Editor | • Activity properties that are displayed to the user.  
• Form properties that are displayed to the user. |
| Provisioning Team Editor | • Provisioning team display name and descriptions |

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 i18nLocalizedMapResolver 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.

2.10.1 Supported Languages

You can localize the display labels in any language displayed in the localization dialog box. The supported languages are defined in the User Application driver. You can modify the list of supported locales as described in Section 2.9, “Specifying Locales and Localization Resource Groups,” on page 31.

2.10.2 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

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

2.10.3 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 a container node or an object in the Provisioning view.
2 Select **Localize > Export Localization Data**.

3 Fill in the fields as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Store in folder</strong></td>
<td>Specify the name of a local folder where the exported files should be written.</td>
</tr>
<tr>
<td><strong>Prefix for generated files</strong></td>
<td>Specify a prefix for the generated files. Determine a naming strategy so you are able to identify the files for projects.</td>
</tr>
<tr>
<td><strong>File Type</strong></td>
<td>Select <strong>XML</strong> or <strong>Properties</strong> depending on the encoding or format you prefer. XML files are UTF-8 encoded. Properties use Unicode*.</td>
</tr>
</tbody>
</table>
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 a container node or an object in the Provisioning view.
2. Select Localize > Import Localization Data.

3. Fill in the fields as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select the languages to export</td>
<td>Select the languages you 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.</td>
</tr>
<tr>
<td>Prompt before overwriting existing files</td>
<td>If this option is selected, Designer prompts you before it overwrites any existing files of the same name in the target directory.</td>
</tr>
</tbody>
</table>
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

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

- Section 3.1, “About the Directory Abstraction Layer,” on page 41
- Section 3.2, “Working with Entities and Attributes,” on page 45
- Section 3.3, “Working with Lists,” on page 53
- Section 3.4, “Working with Queries,” on page 57
- Section 3.5, “Working with Relationships,” on page 60
- Section 3.6, “Working with Configuration Settings,” on page 63
- Section 3.7, “Directory Abstraction Layer Property Reference,” on page 64

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 42) 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.
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.

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 64.

The following table describes the directory abstraction layer toolbar:

<table>
<thead>
<tr>
<th>Node</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entities</strong></td>
<td>Entities represent the Identity Vault objects available to the User Application. There are two types of entities:</td>
</tr>
<tr>
<td></td>
<td>- 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.</td>
</tr>
<tr>
<td></td>
<td>- 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:</td>
</tr>
<tr>
<td></td>
<td>- Used by the Org Chart portlet to determine relationships.</td>
</tr>
<tr>
<td></td>
<td>- Used in the Search List, Create, and Detail portlets to provide selection lists and DN contexts.</td>
</tr>
<tr>
<td></td>
<td>- Available to the workflow request and approval flow forms you define using the provisioning request definition editor.</td>
</tr>
<tr>
<td><strong>Lists</strong></td>
<td>Defines the contents of global lists. Global lists are:</td>
</tr>
<tr>
<td></td>
<td>- Associated with an attribute. The User Application displays the attribute values as a drop-down list in the User Application.</td>
</tr>
<tr>
<td></td>
<td>- Used to display Resource Request categories.</td>
</tr>
<tr>
<td><strong>Queries</strong></td>
<td>Lets you define LDAP search criteria that can be run from a workflow form.</td>
</tr>
<tr>
<td><strong>Relationships</strong></td>
<td>Lets you map hierarchical relationships among schema-based entities. Used by the Organization Chart action of the Identity Self-Service tab of the User Application and in iManager when defining provisioning teams.</td>
</tr>
<tr>
<td><strong>Configuration</strong></td>
<td>General configuration parameters.</td>
</tr>
</tbody>
</table>
Table 3-1 Directory Abstraction Layer Toolbar

<table>
<thead>
<tr>
<th>Toolbar Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Add Entity Wizard" /></td>
<td>Launches the Add Entity Wizard.</td>
</tr>
<tr>
<td><img src="image" alt="Add Attribute Wizard" /></td>
<td>Launches the Add Attribute Wizard.</td>
</tr>
<tr>
<td><img src="image" alt="New List Wizard" /></td>
<td>Launches the New List Wizard.</td>
</tr>
<tr>
<td><img src="image" alt="New Query Wizard" /></td>
<td>Launches the New Query Wizard</td>
</tr>
<tr>
<td><img src="image" alt="New Relationship Wizard" /></td>
<td>Launches the New Relationship Wizard.</td>
</tr>
<tr>
<td><img src="image" alt="Set Global Access Modifiers" /></td>
<td>Launches the Set Global Access Modifiers dialog box.</td>
</tr>
<tr>
<td><img src="image" alt="Set Global Localization" /></td>
<td>Launches the Set Global Localization dialog box.</td>
</tr>
<tr>
<td><img src="image" alt="Expand/Collapse" /></td>
<td>Expands and collapses the directory abstraction layer tree.</td>
</tr>
</tbody>
</table>

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

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

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**

![Diagram showing DAL User Application Default Entities](image)

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 46 and Section 3.2.3, “Adding Attributes,” on page 51. 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 26.

#### 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 46.

- **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 73.

### 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 want to create a directory of employees; one for a public site and one for an internal site. On the public site you want to supply first and last names and a phone number, but on the internal site, you want 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 the Portlet Reference section 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.
2 Fill in the fields as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity Manager Project and</td>
<td>The Identity Manager project and the provisioning application where you want to add the entity and attributes.</td>
</tr>
<tr>
<td>Provisioning Application</td>
<td></td>
</tr>
<tr>
<td>NOTE: These fields display when</td>
<td>you launch the wizard from the File menu.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Entity Key</td>
<td>A unique identifier for the entity.</td>
</tr>
<tr>
<td>Display Label</td>
<td>The string displayed when the entity is displayed by the User Application. You can localize this label. For more information, see Section 2.10, “Localizing Display Labels,” on page 36.</td>
</tr>
</tbody>
</table>
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 53.

5 Click Finish.

The property page displays for editing. For more information, see “Entity Properties” on page 64. 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 box by adding a filter. To add a filter:

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

   ![Class Filters dialog box](image)

   By default, Designer does not apply any class filters. The Class Filter dialog box contains two predefined 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:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td>Choose one of the string comparison operators, such as contains, starts-with, ends-with, then type the string to compare against.</td>
</tr>
</tbody>
</table>

   ![Filter buttons](image)
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 53.

4 Click OK. The property page displays for editing.
For more information, see “Attribute Properties” on page 68. 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 51 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>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>Click <em>Build ECMAScript Expression</em> 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 267.</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>List Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delegate Relationship</td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| Identity Manager Project and Provisioning Application | 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 File menu. |
| List Key                         | The unique identifier for the list.                                         |
| Display Label                    | The string used when the list is displayed in the User Application. You can localize this label. For more information, see Section 2.10, "Localizing Display Labels," on page 36. |
3 Click *Finish*. The Global Lists property page displays for editing.

4 Fill in the fields as follows:

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

The following table describes the wizard’s buttons:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="+" /></td>
<td>Adds a new value</td>
</tr>
<tr>
<td><img src="image" alt="↑" /></td>
<td>Moves the row up or down in the list. This order specifies how the labels are displayed in the User Application.</td>
</tr>
<tr>
<td><img src="image" alt=" Localization" /></td>
<td>Displays the localization dialog box. For more information on using the dialog box, see Section 2.10, &quot;Localizing Display Labels,&quot; on page 36.</td>
</tr>
<tr>
<td><img src="image" alt="X" /></td>
<td>Deletes the row.</td>
</tr>
</tbody>
</table>

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 by 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.

2. Fill in the fields as follows:
Click **Finish**.

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

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

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*.

<table>
<thead>
<tr>
<th>Parameter Keys</th>
<th>Parameter Display Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>param1</td>
<td>label</td>
</tr>
</tbody>
</table>

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 289.

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*).

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 box.

![Predefined Parameters](image)

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 77.

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 Create 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.

2 Fill in the fields as follows:
For property definitions, see Section 3.7.4, “Relationship Properties,” on page 78.

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:

![Manager-Employee(region)](image)

Access
Specify the type of application which can use this relationship:
- [ ] Used by Organizational Chart
- [ ] Used by Team-Management

Enable Cascading Relationship
Maximum levels to cascade

Relationship
Provide a user-friendly label for the relationship:
- [ ] Key: User
- [ ] Display Label: Manager-Employee(region)

Source Object
Target Object
- [ ] Source Entity: User
- [ ] Target Entity: User
- [ ] Source Attribute: This entry's key
- [ ] Target Attribute: Manager

Conditions
Specify any additional conditions for your target attribute:
- Region equals Latin

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>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default ‘My Profile’ Entity</td>
<td>Defines the entity to display when the user clicks My Profile in the user interface.</td>
</tr>
<tr>
<td></td>
<td>This field is restricted to show only entities whose object class is user (or LDAP inetOrgPerson).</td>
</tr>
<tr>
<td>Default LDAP Naming Attributes</td>
<td>Defines the default LDAP naming attribute if the entity’s Create Naming Attribute is not defined.</td>
</tr>
</tbody>
</table>
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 64
- Section 3.7.2, “Attribute Properties,” on page 68
- Section 3.7.3, “Queries Properties,” on page 77
- Section 3.7.4, “Relationship Properties,” on page 78

### 3.7.1 Entity Properties

You can set the following kinds of properties on entities:

- “Entity Access Properties” on page 65
- “Entity General Properties” on page 65
- “Entity Auxiliary Properties” on page 65
- “Entity Search Properties” on page 66
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**

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

Entity General Properties

**Table 3-7  Entity General Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>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.</td>
</tr>
<tr>
<td>Display Label</td>
<td>Defines how the object is shown in the user interface.</td>
</tr>
<tr>
<td>Class Name</td>
<td>The eDirectory object class name.</td>
</tr>
<tr>
<td>LDAP Name</td>
<td>The LDAP object class name.</td>
</tr>
<tr>
<td>Include in Search</td>
<td>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).</td>
</tr>
</tbody>
</table>

Entity Auxiliary Properties
### Table 3-8  Entity Auxiliary Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Auxiliary Classes | A list of zero or more auxiliary classes for this entity. If you are adding auxiliary classes, you are prompted to define:  
  - The auxiliary class by selecting from the list of those available  
  - Whether it is searchable. Setting searchable to True applies a filter to LDAP searches that involve directory abstraction layer relationships. For example, if you added an aux class to the user entity and specified that the aux class was searchable, the Org Chart (using the manager-employee relationship) would display only the employees that have the aux class.  
  - Whether to Add Always. 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**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Search Container | The distinguished name of the LDAP node or container where searching starts (the search root). For example: ou=sample,o=ourOrg  
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 67. |
| Search Scope | Specifies where the search occurs in relation to the search root. Values are:  
<Default>: This search scope is the same as selecting Containers and subcontainers.  
Container: The search occurs in the search root DN and all entries at the search root level.  
Container and subcontainers: The search occurs in the search root DN and all subcontainers. This is the same as selecting <Default>.  
Object: Limits the search to the object specified. This search is used to verify the existence of the specified object. |
| 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. |
### Entity Create Properties

**Table 3-10  Entity Create Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Container</td>
<td>The name of the container where a new entity of this type is created. 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 67. 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.</td>
</tr>
<tr>
<td>Create Naming Attribute</td>
<td>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.</td>
</tr>
<tr>
<td>LDAP attribute</td>
<td>The LDAP attribute for the Create Naming Attribute.</td>
</tr>
<tr>
<td>Create Naming Label</td>
<td>Display label displayed in the User Application for the Create Naming Attribute.</td>
</tr>
</tbody>
</table>

### Entity Password Management Properties

**Table 3-11  Entity Password Management Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password required when entity is created</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Using Predefined Parameters

The directory abstraction layer editor allows you to use predefined parameters for certain values.
### Table 3-12  Predefined Parameters

<table>
<thead>
<tr>
<th>Predefined Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%driver-root%</td>
<td>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.</td>
</tr>
<tr>
<td>%user-root%</td>
<td>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.</td>
</tr>
<tr>
<td>%group-root%</td>
<td>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.</td>
</tr>
</tbody>
</table>

#### 3.7.2 Attribute Properties

You can set the following kinds of properties on attributes:

- “Attribute Access Properties” on page 68
- “Attribute General Properties” on page 70
- “Attribute Default Value Properties” on page 70
- “Attribute UI Control Properties” on page 70
- “Attributes and DNLookup Properties” on page 73

#### 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>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit</td>
<td>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.</td>
</tr>
<tr>
<td>Enable</td>
<td>When deselected, this attribute cannot be used by the User Application. It is the same as removing the entry from the file.</td>
</tr>
</tbody>
</table>
Attribute General Properties

Hide
Controls whether the Hide check box in the User Application is enabled or disabled. The Hide check box allows users to control whether an attribute (such as a photo) is displayed by the application.

When deselected, the Hide check box is disabled for this attribute, so the user cannot choose to hide this attribute.

When selected, the Hide check box can be enabled in the User Application. However, the following must also be true of the logged-in user:

- He or she is either the owner of the attribute or a User Application Administrator.
- He or she has Trustee rights to update the srvprvHideAttributes attribute on the Identity Vault.

If these requirements are not met, then the Hide check box is disabled in the user interface even if this setting is selected (True).

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.

The Search and Hide properties are mutually exclusive. If Hide is selected (True), Search cannot also be selected (True). If Search is selected (True), Hide cannot be selected (True).

Multivalue
Specifies whether this attribute can be multivalued, for example, a phone number.

When selected, the attribute can be multivalued.

Read
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.

Require
When this option is selected, the attribute must be supplied.

Search
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.

TIP: If an attribute used in a search is also indexed in eDirectory, the search is faster.

The Search and Hide properties are mutually exclusive. If Hide is selected (True), Search cannot also be selected (True). If Search is selected (True), Hide cannot be selected (True).

View
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.

Attribute General Properties

Table 3-14  Attribute General Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>The unique identifier for the attribute.</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Type</td>
<td>Choose a data type from the following list:</td>
</tr>
<tr>
<td></td>
<td>• Binary</td>
</tr>
<tr>
<td></td>
<td>• Boolean</td>
</tr>
<tr>
<td></td>
<td>• DN</td>
</tr>
<tr>
<td></td>
<td>• Integer</td>
</tr>
<tr>
<td></td>
<td>• LocalizedString</td>
</tr>
<tr>
<td></td>
<td>• String</td>
</tr>
<tr>
<td></td>
<td>• Time</td>
</tr>
</tbody>
</table>
Format Types are dependent on the data type. For example, a Time data type can only be associated with Date and DateTime formats.
Control Type Types include:

**DNLookup:** Defines that this attribute contains a DN reference. Use when you want 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.
- 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.

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.

For more information on defining this property, see the "Attributes and DNLookup Properties" on page 73. 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 154.

**Global List:** Display this attribute as a drop-down list whose contents are defined in a file outside of this attribute definition. Click *Go to list* to access the Global List editor for the selected list.

For more information, see Section 3.3, "Working with Lists," on page 53.

**Local List:** Display this attribute as a drop-down list whose contents are defined with this attribute. To define a local list:

1. With the attribute selected, set the control type to *Local List*.

   ![Local List](image)

   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.
      
      In the *Value* column, type the value to write to the Identity Vault. It can include letters, numbers, and the underscore (_) character.
      
      3. In the *Labels* column, type the text you want displayed in the user interface.

**Range:** 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.
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 that 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:
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:

![Group DNLookup Definition](image)

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:
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 3-5** Search Criteria

![Search Criteria](image)

**Figure 3-6** Object Lookup

![Object Lookup](image)

Select an object from the list:

**Description**
- Accounting
- Executive Management
- Human Resources
- Improve Customer Service task force
- Information Technology
- Marketing
- Sales

1 - 7 of 7
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**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup Entity</td>
<td>The name of the entity to search. For example, suppose that the User entity contains an attribute for Manager. To populate that field, you’d need to know which users are Managers.</td>
</tr>
<tr>
<td>Lookup Attributes</td>
<td>Choose one or more attributes to display when a search is performed.</td>
</tr>
<tr>
<td>Perform Automatic Query</td>
<td>Defines how the Lookup Attributes are displayed.</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detail entity</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Attributes to Update</td>
<td>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.</td>
</tr>
<tr>
<td>Target Attributes to Update</td>
<td>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.</td>
</tr>
<tr>
<td>Target Auxiliary Classes Needed, if any</td>
<td>Name of the auxiliary class that contains the Target Attributes to Update.</td>
</tr>
</tbody>
</table>

3.7.3 Queries Properties

You can set the following kinds of Queries properties:

- “Queries General Properties” on page 77
- “Query Parameters Properties” on page 78
- “Query Search Properties” on page 78

Queries General Properties

Table 3-19  Queries General Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>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.</td>
</tr>
<tr>
<td>Query Entity</td>
<td>Select an entity from the drop-down list box. The resulting LDAP search is on this entity.</td>
</tr>
<tr>
<td>Display Label</td>
<td>Type a string to display in the directory abstraction layer editor and Provisioning view. This value is not visible in the Expression Builder.</td>
</tr>
</tbody>
</table>
Query Parameters Properties

Table 3-20  Queries Parameters Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Keys</td>
<td>A unique identifier for the key. You pass this key when calling the</td>
</tr>
<tr>
<td></td>
<td>globalQuery() method on a form.</td>
</tr>
</tbody>
</table>

| Parameter Display Labels | 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

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

3.7.4 Relationship Properties

Relationship properties include:

- “Relationship Access Properties” on page 79
- “Relationship Properties” on page 79
Relationship Access Properties

Table 3-22  Relationship Access Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used by Organizational Chart</td>
<td>When selected, this relationship can be used by the Org Chart portlet.</td>
</tr>
<tr>
<td>Used by Team Management</td>
<td>When selected, this relationship can be used to define the provisioning team members in iManager.</td>
</tr>
<tr>
<td></td>
<td>For example, if Used by Team Management 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.</td>
</tr>
<tr>
<td></td>
<td>If Enable Cascading Relationship is selected, then the team can include several levels within the organization. You define the number of levels via Maximum Levels to Cascade.</td>
</tr>
</tbody>
</table>

Relationship Properties

Table 3-23  Relationship Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>The read-only unique identifier for the relationship.</td>
</tr>
<tr>
<td></td>
<td><strong>TIP:</strong> You specify this value in the Org Chart Portlet preference sheet.</td>
</tr>
<tr>
<td>Display Label</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Click Localize to provide the translation for the display label text.</td>
</tr>
<tr>
<td>Source Entity</td>
<td>Choose an entity from the drop-down list.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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).</td>
</tr>
<tr>
<td>Source Attribute</td>
<td>Choose an attribute from the drop-down list.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Target Entity</td>
<td>Choose an entity for the child object in the hierarchy. In a Manager-Employee relationship, it is user. This entity must contain the attribute that is related to the Source attribute.</td>
</tr>
<tr>
<td>Target Attribute</td>
<td>Choose the attribute that matches the Source Attribute. 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 Source Attribute above), then a relationship can be established.</td>
</tr>
</tbody>
</table>

**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.
Working with the Provisioning Request Definition Editor

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 81
- Section 4.2, “Basic Steps for Creating a Provisioning Request Definition,” on page 87
- Section 4.3, “Guidelines for Creating Workflows,” on page 88
- Section 4.4, “Working with the Installed Templates,” on page 94
- Section 4.5, “Debugging a Workflow,” on page 96

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 94.

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 83
- “Step 2: Approving the Request” on page 83
- “Step 3: Fulfilling the Request” on page 86
- “Step 4: Completing the Workflow” on page 87

**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.
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

<table>
<thead>
<tr>
<th>Process Components</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>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. In a workflow diagram, the activities are represented by boxes. 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 My Tasks in the My Work group of actions. To see which workflow activities have been processed for a particular task, the user can select the task and click the View Comment History button on the Task Detail form. To see which workflow activities have been processed for a particular provisioning request, the user can click My Requests, select the request, and click the View Comment and Flow History button on the Request Detail form. For more information on the My Tasks and My Requests actions, see the Identity Manager 3.5.1 User Application: User Guide.</td>
</tr>
<tr>
<td>Flow paths</td>
<td>Flow paths tie the activities in a workflow together. A flow path represents a path to be followed between two activities. 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. In a workflow diagram, the flow paths are represented by arrows.</td>
</tr>
</tbody>
</table>

**Start activity:** The workflow process begins with the execution of the Start activity. This activity displays the initial request form to the user. After 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 his or her 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 83, “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
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. 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.

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.

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 83, “Approval Condition” is an example of a Condition activity.

The Condition activity supports three possible outcomes or exit paths:

- True
- False
- Error

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>
<thead>
<tr>
<th>Task</th>
<th>Action</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: Use the wizard to create the provisioning request definition</td>
<td>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.</td>
<td>See Chapter 5, “Creating a Provisioning Request Definition,” on page 99.</td>
</tr>
<tr>
<td></td>
<td>Provisioning request definitions are stored locally in the Provisioning\AppConfig\RequestDefs directory within your workspace.</td>
<td></td>
</tr>
<tr>
<td>Step 2: Create the forms</td>
<td>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.</td>
<td>See Chapter 6, “Creating Forms for a Provisioning Request Definition,” on page 109.</td>
</tr>
<tr>
<td>Step 3: Create the workflow diagram</td>
<td>Add the activities to the workflow diagram and connect them with flow paths.</td>
<td>See Chapter 7, “Creating the Workflow for a Provisioning Request Definition,” on page 165</td>
</tr>
</tbody>
</table>
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 88
- Section 4.3.2, “Rules for Flow Paths,” on page 89
- Section 4.3.3, “Understanding Workflow Data,” on page 90

NOTE: You can validate a provisioning request definition before you deploy it. For more information, see Section 2.6, “Validating Provisioning Objects,” on page 27.

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
  - Log activity
  - Branch activity
  - Merge activity
  - Condition activity
  - Mapping activity
  - Workflow Status
  - E-Mail activity
  - Integration
  - Entitlement
  - Entity

- 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>
<thead>
<tr>
<th>Activity</th>
<th>Inbound Paths</th>
<th>Outbound Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>1 (Must always be forward.)</td>
</tr>
<tr>
<td>Approval</td>
<td>1 to n</td>
<td>1 to 5 (Approved, denied, refused, timedout, or error.)</td>
</tr>
<tr>
<td>Log</td>
<td>1 to n</td>
<td>1 (Must always be forward.)</td>
</tr>
<tr>
<td>Branch</td>
<td>1 to n</td>
<td>1 to n</td>
</tr>
<tr>
<td>Merge</td>
<td>1 to n</td>
<td>1 (Must always be forward.)</td>
</tr>
<tr>
<td>Condition</td>
<td>1 to n</td>
<td>2 to 3 (True and false are required; error is optional.)</td>
</tr>
<tr>
<td>Mapping</td>
<td>1 to n</td>
<td>1</td>
</tr>
<tr>
<td>Workflow Status</td>
<td>1 to n</td>
<td>1 (Must always be forward)</td>
</tr>
<tr>
<td>E-Mail</td>
<td>1 to n</td>
<td>1 (Must always be forward)</td>
</tr>
<tr>
<td>Finish</td>
<td>1 to n</td>
<td>0</td>
</tr>
<tr>
<td>Integration</td>
<td>1 to n</td>
<td>1 to 4 (Success, timedout, error, fault)</td>
</tr>
<tr>
<td>Entitlement</td>
<td>1 to n</td>
<td>1 Must always be forward.</td>
</tr>
<tr>
<td>Entity</td>
<td>1 to n</td>
<td>1 Must always be forward.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Activity</th>
<th>Forward</th>
<th>Approved</th>
<th>Denied</th>
<th>Refused</th>
<th>Timedout</th>
<th>True</th>
<th>False</th>
<th>Error</th>
<th>Succes</th>
<th>Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Source</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Approval</td>
<td>Target</td>
<td>Source/Target</td>
<td>Source/Target</td>
<td>Source/Target</td>
<td>Target/Target</td>
<td>Target/Target</td>
<td>Target/Target</td>
<td>Target/Target</td>
<td>Success</td>
<td>Fault</td>
</tr>
<tr>
<td>Log</td>
<td>Source/Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Success</td>
<td>Fault</td>
</tr>
<tr>
<td>Branch</td>
<td>Source/Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Success</td>
<td>Fault</td>
</tr>
<tr>
<td>Merge</td>
<td>Source/Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Success</td>
<td>Fault</td>
</tr>
<tr>
<td>Condition</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Source/Target</td>
<td>Source/Target</td>
<td>Source/Target</td>
<td>Success</td>
<td>Fault</td>
</tr>
<tr>
<td>Mapping</td>
<td>Source</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Success</td>
<td>Fault</td>
</tr>
<tr>
<td>Workflow Status</td>
<td>Source/Target</td>
<td>Target</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E-Mail</td>
<td>Source/Target</td>
<td>Target</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finish</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Success</td>
<td>Fault</td>
</tr>
<tr>
<td>Integration</td>
<td>Source/Target</td>
<td>Source/Target</td>
<td>Source/Target</td>
<td>Source/Target</td>
<td>Source/Target</td>
<td>Source/Target</td>
<td>Source/Target</td>
<td>Source/Target</td>
<td>Success</td>
<td>Fault</td>
</tr>
<tr>
<td>Entitlement</td>
<td>Source/Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Success</td>
<td>Fault</td>
</tr>
<tr>
<td>Entity</td>
<td>Source/Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Target</td>
<td>Success</td>
<td>Fault</td>
</tr>
</tbody>
</table>

### 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. After 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**

<table>
<thead>
<tr>
<th>Data object</th>
<th>Lifetime</th>
<th>Editable</th>
<th>Creator</th>
</tr>
</thead>
<tbody>
<tr>
<td>process</td>
<td>Workflow</td>
<td>No</td>
<td>System</td>
</tr>
<tr>
<td>activities</td>
<td>Workflow</td>
<td>No</td>
<td>System</td>
</tr>
<tr>
<td>activity forms</td>
<td>Activity</td>
<td>Yes</td>
<td>System and workflow designer</td>
</tr>
<tr>
<td>flowdata</td>
<td>Workflow</td>
<td>Yes</td>
<td>Workflow designer</td>
</tr>
</tbody>
</table>

**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**

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

\[
\text{Number}(\text{flowdata.get('cost')}) + \text{Number}(\text{flowdata.get('tax')})
\]

### 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>
<thead>
<tr>
<th>Source Form Field</th>
<th>Target Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acct_Pay</td>
<td>flowdata.SystemAccess/AcctPay</td>
</tr>
<tr>
<td>Acct_Rec</td>
<td>flowdata.SystemAccess/AcctRec</td>
</tr>
</tbody>
</table>

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 workflows containing many forms and data items. To retrieve data from these hierarchies, the following syntax would be used:

\[
\text{flowdata.get('SystemAccess/AcctPay')}
\]

For complete details on building ECMAScript expressions, see Chapter 10, “Working with ECMA Expressions,” on page 267.

### 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:

\[
\text{flowdata.my_ACONTROL}
\]

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

\[
\text{flowdata.get('my_ACONTROL')}
\]

\[
\text{flowdata.getObject('my_ACONTROL')}
\]

### 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>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self Provision Approval</td>
<td>Allows a provisioning request to be fulfilled without any approvals.</td>
</tr>
<tr>
<td>One Step Approval (Timeout Approves)</td>
<td>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.</td>
</tr>
<tr>
<td>Two Step Sequential Approval (Timeout Approves)</td>
<td>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.</td>
</tr>
<tr>
<td>Three Step Sequential Approval (Timeout Approves)</td>
<td>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. This template supports sequential processing.</td>
</tr>
<tr>
<td>Template</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Four Step Sequential Approval (Timeout Approves)</td>
<td>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. This template supports sequential processing.</td>
</tr>
<tr>
<td>Five Step Sequential Approval (Timeout Approves)</td>
<td>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. This template supports sequential processing.</td>
</tr>
<tr>
<td>One Step Approval (Timeout Denies)</td>
<td>Requires a single approval for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</td>
</tr>
<tr>
<td>Two Step Sequential Approval (Timeout Denies)</td>
<td>Requires two approvals for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</td>
</tr>
<tr>
<td>Three Step Sequential Approval (Timeout Denies)</td>
<td>Requires three approvals for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</td>
</tr>
<tr>
<td>Four Step Sequential Approval (Timeout Denies)</td>
<td>Requires four approvals for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</td>
</tr>
<tr>
<td>Five Step Sequential Approval (Timeout Denies)</td>
<td>Requires five approvals for the provisioning request to be fulfilled. If an activity times out, the workflow denies the request.</td>
</tr>
<tr>
<td>Two Step Parallel Approval (Timeout Approves)</td>
<td>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 parallel processing.</td>
</tr>
<tr>
<td>Three Step Parallel Approval (Timeout Approves)</td>
<td>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. This template supports parallel processing.</td>
</tr>
</tbody>
</table>
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 200.

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.
Creating a Provisioning Request Definition

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

- Section 5.1, “About the Wizard and the Overview Tab,” on page 99
- Section 5.2, “Using the Wizard to Create a Provisioning Request Definition,” on page 101
- Section 5.3, “Using the Overview Tab to Modify Basic Settings,” on page 105

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:

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

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.

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 105).

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 102
- Section 5.2.2, “From Concept to Finished Product,” on page 104

5.2.1 Using a Template

To create a provisioning request definition by 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 99). In step 4 on page 101, 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 must 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.  

5 Use the ECMA expression builder to replace the placeholder expression with an expression that specifies the entitlement that you want 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 109. 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.  

**Templates and Flow Strategy**

By default, provisioning request definition templates use the Single Flow flow strategy. The templates assume the recipient is a user DN. If you change the flow strategy you must modify the template. If you change the flow strategy to:

- **Flow per member**: Remove the recipient reference from the request form.
- **Single flow provision members**: Remove the recipient from the request form and add logic to determine the addressee for the first approval activity. The templates assume the recipient is a user DN so you must determine if the recipient is a user or group DN. You can use the IDVault isGroup(String dn) method to determine if the DN is a group. If the recipient is a group DN, you must provide logic for assigning the approval activity addressee.
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 99). In step Step 4 on page 101, 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 116.

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 165.

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 189.

5. Configure the flowpaths between workflow activities.

   To configure flowpaths, see Section 7.4, “Configuring Flow Paths,” on page 174.
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 following table describes each property that you can configure on the Overview tab.

Table 5-1  Overview Properties

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier (CN)</td>
<td>Displays the CN (common name) of the provisioning request definition. The CN cannot be changed.</td>
</tr>
<tr>
<td>Display Name</td>
<td>Specifies the display name of the provisioning request definition. This is the name that is displayed to the user in Designer and Identity Manager.</td>
</tr>
<tr>
<td>Description</td>
<td>Specifies the description of the provisioning request definition.</td>
</tr>
<tr>
<td>Category</td>
<td>Specifies the category to which the provisioning request definition belongs from the list of Provisioning Categories defined in the directory abstraction layer. The Provisioning view displays the provisioning request definitions by categories.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies the status of the provisioning request definition:</td>
</tr>
<tr>
<td><strong>Active:</strong></td>
<td>Select this option to make the provisioning request definition available for use in the User Application.</td>
</tr>
<tr>
<td><strong>Inactive:</strong></td>
<td>Select this option 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.</td>
</tr>
<tr>
<td><strong>Template:</strong></td>
<td>Select this option 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.</td>
</tr>
<tr>
<td><strong>Retired:</strong></td>
<td>Select this option 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.</td>
</tr>
<tr>
<td>Flow Strategy</td>
<td>Specifies the flow strategy for the provisioning request definition:</td>
</tr>
<tr>
<td><strong>Single Flow:</strong></td>
<td>This strategy allows one workflow with one recipient.</td>
</tr>
<tr>
<td><strong>Flow per member:</strong></td>
<td>This strategy allows the recipient to be a group DN. If you select this strategy, the User Application starts a workflow instance for each member of the group, and each workflow can be approved or denied separately. For example, if there is a provisioning request definition for the recipient Human Resources. The Human Resource group has the members ablake and kchester. The User Application passes the Human Resources DN to the provisioning start. The provisioning interface starts two workflow instances; one for ablake and one for kchester.</td>
</tr>
<tr>
<td><strong>Single Flow Provision Members:</strong></td>
<td>This strategy allows the recipient to be a group DN. If you select this strategy, the User Application starts a single workflow for the group. The workflow spawns multiple provisioning steps (one for each member) within the single workflow. The workflow is approved or denied to the group as a whole not for each individual member.</td>
</tr>
<tr>
<td>Notify participants by E-Mail</td>
<td>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 Requests and Approvals tab in the User Application for notifications about tasks.</td>
</tr>
<tr>
<td>Restrict View</td>
<td>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 selected) is that the user can see any requests that the user initiated or for which the user is the recipient.</td>
</tr>
</tbody>
</table>
### Generate Comments
Specifies that the workflow engine should generate comments as the workflow progresses. These comments can be displayed by clicking the following:

- **View Comment and Flow History** in a Request Detail form in the User Application
- **View Comment History** on a Task Detail form in the User Application

### Set Default Completion Status to Approved
Specifies that the default completion status of the provisioning request is approved if selected, or denied if not selected. 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.

### Trustee Rights
Specifies the users and groups that can use the provisioning request definition.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate Comments</td>
<td>Specifies that the workflow engine should generate comments as the workflow progresses. These comments can be displayed by clicking the following:</td>
</tr>
<tr>
<td>Set Default Completion</td>
<td>Specifies that the default completion status of the provisioning request is approved if selected, or denied if not selected. 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.</td>
</tr>
<tr>
<td>Trustee Rights</td>
<td>Specifies the users and groups that can use the provisioning request definition.</td>
</tr>
</tbody>
</table>
You can set these access the properties available on the Overview panel for one or more provisioning request definitions.

1. With the Provisioning view open, select one or more provisioning request definitions, right-click, then select Properties. The following screen displays.

   ![Set properties for multiple Provisioning Request Definitions](image)

2. To modify a property for all of the selected provisioning request definitions, make that change.

3. If one of the properties does not contain a value, the corresponding field is blank. If you modify that field, then the value you select is applied to all of the selected provisioning request definitions.

4. For Trustees, you can delete, add, or merge the values for the selected provisioning request definitions. To merge the values, select the `<`. This adds all of the trustees for any selected provisioning request definition to all of the selected provisioning request definitions.
Creating Forms for a Provisioning Request Definition

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 109
- Section 6.2, “About the Forms Tab,” on page 112
- Section 6.3, “Creating Forms,” on page 116
- Section 6.4, “Action Reference,” on page 125
- Section 6.5, “Form Control Reference,” on page 127
- Section 6.6, “Working with Distinguished Names,” on page 153
- Section 6.7, “Using DAL Queries in Forms,” on page 156
- Section 6.8, “Printing Forms,” on page 161
- Section 6.9, “Providing Direct Access to a Form,” on page 162

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. Forms are also available directly from a URL. For details on this technique, see Section 6.9, “Providing Direct Access to a Form,” on page 162.

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.
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*.

**About Approval Forms**

You can define multiple approval forms for a provisioning request definition, but only one form per *Approval Activity*. You can specify the approval form to associate with an approval activity in the
properties for the activity. You can create an approval form via the *Forms* tab or from the approval activities property sheet.

*Figure 6-3* Sample Resource Approval Form

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 workflow 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 171.

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 127.

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.
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 280. 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 123.

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

- Section 6.2.1, “About Form Selection,” on page 113
- Section 6.2.2, “About Form Controls,” on page 114

### 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>
<thead>
<tr>
<th><strong>Table 6-1  Form Selection Toolbar Options</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Button</strong></td>
</tr>
<tr>
<td>[ ]</td>
</tr>
<tr>
<td>[ ]</td>
</tr>
</tbody>
</table>
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.

```
<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Control Type</th>
<th>Layout Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>string</td>
<td>text</td>
<td>1</td>
</tr>
<tr>
<td>Phone</td>
<td>string</td>
<td>text</td>
<td>2</td>
</tr>
<tr>
<td>Email</td>
<td>string</td>
<td>text</td>
<td>3</td>
</tr>
</tbody>
</table>
```

For information about adding controls, see Section 6.3, “Creating Forms,” on page 116. For more information about individual form controls, see Section 6.5, “Form Control Reference,” on page 127.

- **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.
For more information about the supported actions, see Section 6.4, “Action Reference,” on page 125.

* 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 by 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 123.
6.3 Creating Forms

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

- Section 6.3.1, “Creating New Forms,” on page 116
- Section 6.3.2, “Adding Form Controls and Actions,” on page 117
- Section 6.3.3, “Defining Events,” on page 119
- Section 6.3.4, “Using the Scripts Tab,” on page 123

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.

![New Form Wizard](image)

3 Fill in the fields as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Form Name</strong></td>
<td>Type the name of the form as you want it to appear in Designer.</td>
</tr>
<tr>
<td><strong>Create a form using one of the templates</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

4 Click *OK* to save the form or click *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.
**NOTE:** The Designer places form controls on the form from top to bottom and left to right. Use Linebreaks to force spacing between controls.

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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Form Field Name</strong></td>
<td>A unique name for the field. The name is used in</td>
</tr>
<tr>
<td></td>
<td>- The <em>Workflow</em> tab’s <em>Data Item Mapping</em> dialog box.</td>
</tr>
<tr>
<td></td>
<td>- The <em>ECMA expression builder</em> dialog box.</td>
</tr>
<tr>
<td></td>
<td>- An internal XML reference in the provisioning request definition file.</td>
</tr>
<tr>
<td><strong>Data Type</strong></td>
<td>The field’s data type. The data type determines the valid control types and the type of validation performed.</td>
</tr>
<tr>
<td><strong>Control Type</strong></td>
<td>The type of visual control used to display or edit the data. The selection list is filtered based on the selected data type.</td>
</tr>
<tr>
<td><strong>Linebreaks</strong></td>
<td>Defines the number of lines you want inserted after the control.</td>
</tr>
</tbody>
</table>

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 127.

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.

   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:
6 Save the form.

### 6.3.3 Defining Events

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

- “Defining an Event” on page 119
- “Creating Custom Events” on page 120

#### Defining 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`.

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 120.

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 129.

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 drop-down list are populated from a query launched by the `namechange` custom event.
The Name field defines an Events property that fires the namechange event on an onchange event. The definition is shown in Figure 6-8.
Figure 6-8  Sample field.FireEvent() Method

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

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Action Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>onchange</td>
<td><code>field.fireEvent(&quot;namechange&quot;)</code></td>
</tr>
</tbody>
</table>
For more information on using queries, see Section 6.7, “Using DAL Queries in Forms,” on page 156.

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>
<thead>
<tr>
<th>Script type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>external</td>
<td>The script is incorporated into the page by reference using the supplied URL. The script block will look something like this: <code>&lt;script type=&quot;test/javascript&quot; scr=&quot;http://some.server/custom.js&quot;&gt;</code>. The custom.js file is imported at form load.</td>
</tr>
<tr>
<td>inline</td>
<td>The script is inserted directly into the form in a <code>&lt;script&gt;</code> block.</td>
</tr>
</tbody>
</table>

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:

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

To create an inline script:

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Specify a meaningful name. This value displays in the Event Action Expression Builder.</td>
</tr>
<tr>
<td>Type</td>
<td>Set the value to <code>inline</code>.</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Form Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApprovalAction</td>
<td>Approval</td>
<td>Causes the Approval activity to complete and follow the approved 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. <strong>TIP:</strong> An ApprovalAction requires the Approval Activity associated with the form to have an approved flow path exiting the activity.</td>
</tr>
<tr>
<td>CancelAction</td>
<td>Request and Approval</td>
<td>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.</td>
</tr>
</tbody>
</table>
CommentAction | Approval forms | Generates a button with the default label set to View Comment History. 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.

| Task Comments |
|---|---|---|---|
| Date | Activity | User | Comments |
| 06/01/2006 09:20:48 AM | Start | System | Workflow Started |
| 06/01/2006 09:20:48 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 |

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.

DenyAction | Approval | 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.

TIP: A DenyAction requires the Approval Activity associated with the form to have a deny flow path exiting the activity.

RefusalAction | Approval | 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.

TIP: A RefusalAction requires the Approval Activity associated with the form to have a refusal flow path exiting the activity.

SubmitAction | Request and Approval | 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.
The following table describes the properties you can set on actions.

**Table 6-4  Action Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Label</td>
<td>Specifies the text to display on the button.</td>
</tr>
<tr>
<td>Visible</td>
<td>If True, specifies whether the action is visible at runtime.</td>
</tr>
<tr>
<td>Block On Error</td>
<td>If True, specifies that the action is blocked if any of the form’s controls fail validation. This is recommended for the SubmitAction. Do not set to False if the action button submits data; otherwise, invalid data can be submitted, causing unexpected results.</td>
</tr>
<tr>
<td>Hide If Read Only</td>
<td>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 ApprovalAction, DenyAction, or RefusalAction, this property must be set to true. If it is set to False, you encounter a validation error and cannot deploy the form.</td>
</tr>
</tbody>
</table>

### 6.5 Form Control Reference

This section describes the controls you can add to a form.
<table>
<thead>
<tr>
<th>Control Type</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Boolean</td>
</tr>
<tr>
<td>CheckBoxPickList</td>
<td>x</td>
</tr>
<tr>
<td>DatePicker</td>
<td></td>
</tr>
<tr>
<td>DateTimePicker</td>
<td></td>
</tr>
<tr>
<td>DNContainer</td>
<td></td>
</tr>
<tr>
<td>DNDisplay</td>
<td></td>
</tr>
<tr>
<td>DNLookup</td>
<td></td>
</tr>
<tr>
<td>DNMaker</td>
<td></td>
</tr>
<tr>
<td>DNQuery</td>
<td></td>
</tr>
<tr>
<td>Global List</td>
<td></td>
</tr>
<tr>
<td>Html</td>
<td></td>
</tr>
<tr>
<td>MVCheckbox</td>
<td>x</td>
</tr>
<tr>
<td>MVEditor</td>
<td>x</td>
</tr>
<tr>
<td>PickList</td>
<td>x</td>
</tr>
<tr>
<td>Static List</td>
<td>x</td>
</tr>
<tr>
<td>Text</td>
<td>x</td>
</tr>
<tr>
<td>Text Area</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td></td>
</tr>
<tr>
<td>TrueFalseRadioButtons</td>
<td>x</td>
</tr>
<tr>
<td>TrueFalseSelectBox</td>
<td></td>
</tr>
</tbody>
</table>

- Section 6.5.1, “Controls for User-Entered Comments,” on page 129
- Section 6.5.2, “General Form Control Properties,” on page 129
- Section 6.5.3, “CheckBoxPickList,” on page 130
- Section 6.5.4, “DatePicker,” on page 131
- Section 6.5.5, “DateTimePicker,” on page 132
- Section 6.5.6, “DNContainer,” on page 133
- Section 6.5.7, “DNDisplay,” on page 134
- Section 6.5.8, “DNLookup,” on page 135
- Section 6.5.9, “DNMaker,” on page 139
- Section 6.5.10, “DNQuery,” on page 140
- Section 6.5.11, “Global List,” on page 141
- Section 6.5.12, “Html,” on page 142
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 \textit{CommentAction} or \textit{UpdateAction}. The comments are not part of the workflow data so you cannot access them via the \texttt{flowdata} object. The comments are special data items stored in the \texttt{afcomment} table of the workflow database. The comments are persisted as long as the row for the \texttt{requestid} in the \texttt{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 \texttt{apwaComment} and the Control Type is \texttt{TextArea}. A single form can contain only one comment field.

2. Add a \textit{CommentAction} or \textit{UpdateAction} to the form.
   For more information, see Section 6.4, “Action Reference,” on page 125.

6.5.2 General Form Control Properties

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

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
Property Name & Description \\
\hline
\textit{Display label} & Specifies the label to display to identify the control. It is localizable. \\
\textit{Editable} & Specifies if the control is editable (True). Otherwise, it displays as read-only. \\
\hline
\end{tabular}
\end{table}
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 check boxes.

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
### CheckboxPickList Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entity Key for DN expression lookup</strong></td>
<td>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. Leave this value blank if you want to display the full DN or CN value retrieved from the Identity Vault. 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.</td>
</tr>
<tr>
<td><strong>Display expression</strong></td>
<td>Required when you specify an Entity Key for DN expression Lookup. Choose the attributes to display as the check box labels. For example, to display the user entity’s first and last name attributes, construct an expression like this: FirstName LastName. The attribute’s directory abstraction layer properties for View, Read, Search, and Required must be set to True.</td>
</tr>
<tr>
<td><strong>Allow multiple selections</strong></td>
<td>When set to True, users can select more than one entry.</td>
</tr>
<tr>
<td><strong>Sort entries</strong></td>
<td>When set to True, sorts results in ascending order. For details, see “Sort Order” on page 130.</td>
</tr>
</tbody>
</table>

### 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 by 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: 05/25/2006 10:47:25 AM |

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:
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 value in a text field. At runtime, the form automatically validates the date and time by 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:
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:

![Sample DateTimePicker Control](image)

![DateTimePicker Calendar Control](image)

### Table 6-9 DateTimePicker Control Properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Day headers</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Field width in pixels</strong></td>
<td>Use this field to configure the field’s visible width on the form. The default is 200 pixels.</td>
</tr>
<tr>
<td><strong>isDateTime</strong></td>
<td>When set to False, the Calendar pop-up does not display the time.</td>
</tr>
<tr>
<td><strong>Month names</strong></td>
<td>A comma-separated, single-quoted list of values displayed by the Calendar pop-up to indicate the month name. This value is localizable.</td>
</tr>
<tr>
<td><strong>Show date picker</strong></td>
<td>When set to True, displays the calendar pop-up. If set to false, the calendar pop-up does not display. Users must type the proper format for the locale when they type the date in the text field.</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity key used for object lookup</td>
<td>Choose an entity from the drop-down list. 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.</td>
</tr>
<tr>
<td>Field width in pixels</td>
<td>Use this field to configure the field’s visible width on the form. The default is 200 pixels.</td>
</tr>
<tr>
<td>Root container</td>
<td>Specify a root container for lookups when users click the Object Selector button.</td>
</tr>
<tr>
<td>Show clear button</td>
<td>If set to True, the form displays the Reset field button</td>
</tr>
<tr>
<td>Show object history button</td>
<td>If set to True, the form displays the Show history button.</td>
</tr>
<tr>
<td>Show object selector button</td>
<td>If set to True, the form displays the Object Selector button.</td>
</tr>
</tbody>
</table>

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
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

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display expression</td>
<td>Leave this value blank if you want to display the full DN or CN value. 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 Entity key for DN expression lookup.) For example, to show the user entity’s first and last name attributes, construct an expression like this: FirstName LastName. 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 68.</td>
</tr>
</tbody>
</table>
| Entity key for DN expression lookup | Leave this value blank if you want to display the full DN or CN value retrieved from the Identity Vault. 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 Display expression property. The entity you choose must:  
  - Have the directory abstraction layer View property set to True.  
  - Be the entity of the DN you are working with. For more information, see Section 6.6, “Working with Distinguished Names,” on page 153. |

The buttons associated with the DNLookup control are described in Table 6-12.
### Table 6-12 DNLookup Control Buttons

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Search" /></td>
<td>Launches an object lookup dialog box. You define whether the dialog box displays containers or objects via the <strong>Object Selector type</strong> property. The following is an example of an object lookup:</td>
</tr>
<tr>
<td><img src="image" alt="History" /></td>
<td>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.</td>
</tr>
<tr>
<td><img src="image" alt="Reset" /></td>
<td>Reset field. Deletes the field contents. The availability of this button is controlled by the Show clear button property.</td>
</tr>
</tbody>
</table>

The attributes shown in the drop-down list (**Description** in the above example) are specified in the directory abstraction layer. The availability of this button is controlled by the Show object selector property.
Table 6-13  DNLookup Control Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Display expression** | 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.

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 *Entity key for DN expression lookup*.)

For example, to show the user entity’s first and last name attributes, construct an expression like this: FirstName LastName.

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 68. |
| **Entity key for DN expression lookup** | 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.

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 *Display expression* property.

The entity you choose must:

- Have the directory abstraction layer View property set to True.
- Be the entity of the DN you are working with. |

For more information, Section 6.6, “Working with Distinguished Names,” on page 153. |
**Object Selector type**

Determines whether the object selector dialog box performs an Object Lookup or a Container Lookup. The following is an example of an Object Lookup:

```
Object Lookup

Search object list: (example: J, L8, ID, *)

FirstName: [input field] [search button]
```

- **paramlist**: Causes the object selector dialog to perform an object lookup. You specify the lookup criteria via the **Entity key used for object lookup** property.

- **container**: 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 **Entity key used for object lookup** property. For example, if the **Entity key used for object lookup** 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.

**Field width in pixels**

Use this field to configure the field’s visible width on the form. The default is 200 pixels.

**Show clear button**

If set to True, the form displays the **Reset field** button.

**Show object history button**

If set to True, the form displays the **Show history** 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*

![Sample DNMaker Control](image)

*Table 6-14  DNMaker Control Buttons*

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Launches an object selector for container searches like the one shown below." /></td>
<td>Launches an object selector for container searches like the one shown below.</td>
</tr>
<tr>
<td><img src="image" alt="The container search root is defined for the entity specified in the Entity used for object lookup property. The availability of this button is controlled by the Show object selector property." /></td>
<td>The container search root is defined for the entity specified in the <em>Entity used for object lookup</em> property. The availability of this button is controlled by the Show object selector property. <em>Show history.</em> 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.</td>
</tr>
<tr>
<td><img src="image" alt="Reset field. Deletes the field contents. The availability of this button is controlled by the Show clear button property." /></td>
<td>Reset field. Deletes the field contents. The availability of this button is controlled by the Show clear button property.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAL global query key</td>
<td>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 156. For more information about defining DAL queries, see Section 3.4, “Working with Queries,” on page 57.</td>
</tr>
</tbody>
</table>
| DAL global query parameter(s) | Specifies the value for the query parameters. For example, this passes the String Sales to the Queries parameter called groupname:  

(function (){return {
"groupname":"Sales"}})();  

Display expression | When you populate the control with initial data from a Data Item Mapping value, use this property to specify the attributes to display. |
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>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity key for DN expression lookup</td>
<td>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. 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 Display expression property. The entity you choose must: • Have the directory abstraction layer View property set to True. • Be the entity of the DN you are working with. For more information, Section 6.6, “Working with Distinguished Names,” on page 153.</td>
</tr>
<tr>
<td>Field width in pixels</td>
<td>Use this field to configure the field’s visible width on the form. The default is 200 pixels.</td>
</tr>
<tr>
<td>Show clear button</td>
<td>If set to True, the form displays the Reset field button</td>
</tr>
<tr>
<td>Show object history button</td>
<td>If set to True, the form displays the Show history button. When a DAL global query is specified, the object history button is not shown.</td>
</tr>
<tr>
<td>Show object selector button</td>
<td>If set to True, the form displays the Object Selector button.</td>
</tr>
</tbody>
</table>

**Table 6-17  Global List Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAL global list key</td>
<td>Specifies the unique identifier of the global list. This must correspond to the key specified in the directory abstraction layer.</td>
</tr>
</tbody>
</table>

For more information about global lists, see Section 3.3, “Working with Lists,” on page 53.
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:

```javascript
(function() { return "<yourTag yourAttr='your attr value' />"; } )();
```

For example:

```javascript
(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 labeled check boxes. You specify the label and its associated values through the List item property. A sample MVCheckbox control is shown below.

![Sample MVCheckbox Control](image)

**Table 6-18 MVCheckbox Control Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List item</td>
<td>Allows you to define a set of static values that comprise the checkbox labels and values. Click the List property button to launch the list value dialog box shown here:</td>
</tr>
</tbody>
</table>

![List Value Dialog Box](image)

**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 278.
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

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Object Lookup" /></td>
<td>Launches a search dialog box called an object selector. The object selector dialog box looks like this:</td>
</tr>
<tr>
<td><img src="image" alt="Show History" /></td>
<td>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.</td>
</tr>
<tr>
<td><img src="image" alt="Reset Field" /></td>
<td>Reset field. Deletes the field contents. The availability of this button is controlled by the Show clear button property.</td>
</tr>
</tbody>
</table>

The user can select a value from the list to populate the control. The attribute displayed in the drop-down list (Description 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 70. The availability of this button is controlled by the Show object selector property.

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

MVEditorControl: Jack Miller

✓ [ ] [ ]
### Table 6-20 MVEditor Control Buttons

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Add" /></td>
<td>Adds an item to the end of the list.</td>
</tr>
<tr>
<td><img src="image" alt="Delete" /></td>
<td>Deletes the selected list item.</td>
</tr>
<tr>
<td><img src="image" alt="Edit" /></td>
<td>Edits the selected list item.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add data entry text field</td>
<td>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 list is displayed. If more than one row of data exists, then the drop-down list always displays.</td>
</tr>
<tr>
<td>DAL Global Query</td>
<td>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 156. For information about defining queries, see Section 3.4, “Working with Queries,” on page 57.</td>
</tr>
<tr>
<td>DAL Global Query Parameter(s)</td>
<td>Specifies the value for the query parameters. For example, this passes the String Sales to the queries parameter called groupname.</td>
</tr>
</tbody>
</table>

```javascript
(function (){return
{"groupname":"Sales"});})();
```
Display expression

Leave this value blank if you want to display the full DN or CN value.

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 Entity key for DN expression lookup.)

For example, to show the user entity’s first and last name attributes, construct an expression like this: FirstName LastName.

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 68.

Enforce uniqueness

Forces user-entered list items to be unique.

Entity key for DN expression lookup

Leave this value blank if you want to display the full DN or CN value retrieved from the Identity Vault.

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 Display expression property.

The entity you choose must

- Have the directory abstraction layer View property set to True.
- Be the entity whose DN you are retrieving from the Identity Vault.


Field width in pixels

Use this field to configure the field’s visible width on the form. The default is 200 pixels.

Ignore case

If set to True, ignore case when enforcing uniqueness.

Lower bound (for numbers only)

Minimum integer or decimal value.

Maximum length

Maximum number of characters for string values. The control blocks input when this value is reached.

Minimum length

Minimum number of characters for string values. The control validates that the user enters at least this number of characters.
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of lines displayed</strong></td>
<td>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. 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 scroll bars. If you set it to 2, Designer generates a warning in the Project Checker view and resets it to 3.</td>
</tr>
<tr>
<td><strong>Numbers only</strong></td>
<td>If set to True, only numbers can be entered.</td>
</tr>
</tbody>
</table>
Object Selector type

Determines whether the object selector dialog box performs an Object Lookup or a Container Lookup. The following is an example of an Object Lookup:

**paramlist**: Causes the object selector dialog box to perform an object lookup. You specify the lookup criteria via the *Entity key used for object lookup* property.

**container**: Causes the object selector 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 *Entity key used for object lookup* property. For example, if the *Entity key used for object lookup* 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.

Resolve DN

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.
**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.

**Property Name** | **Description**
--- | ---
*Root Container* | 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.

*Show object history button* | When set to True, displays the Object History button next to the control.

*Show object selector button* | When set to True, displays the Object Selector button next to the control.

*Sort entries* | When set to True, sorts the results in ascending order. For details, see “Sort Order” on page 130.

*Upper bound (for Numbers only)* | 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 267.
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Display expression</strong></td>
<td>Leave this value blank if you want to display the full DN or CN value.</td>
</tr>
<tr>
<td></td>
<td>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 <em>Entity key for DN expression lookup.</em>)</td>
</tr>
<tr>
<td></td>
<td>For example, to show the user entity’s first and last name attributes, construct an expression like this: <em>FirstName LastName</em>.</td>
</tr>
<tr>
<td></td>
<td>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 68.</td>
</tr>
<tr>
<td><strong>Entity key for DN expression lookup</strong></td>
<td>Leave this value blank if you want to display the full DN or CN value retrieved from the Identity Vault.</td>
</tr>
<tr>
<td></td>
<td>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 <em>Display expression</em> property.</td>
</tr>
<tr>
<td></td>
<td>The entity you choose must:</td>
</tr>
<tr>
<td></td>
<td>• Have the directory abstraction layer View property set to True.</td>
</tr>
<tr>
<td></td>
<td>• Be the entity whose DN you are retrieving from the Identity Vault.</td>
</tr>
<tr>
<td><strong>Field width in pixels</strong></td>
<td>Use this field to configure the field’s visible width on the form. The default is 200 pixels.</td>
</tr>
<tr>
<td><strong>Number of lines displayed</strong></td>
<td>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. The number of lines displayed is related to the <em>Allow multiple selections</em> setting. When <em>Allow multiple selections</em> is set to True, the number of lines displayed is always 3 (or more). When <em>Allow multiple selections</em> 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 scroll bars. If you set it to 2, Designer generates a warning in the Project Checker view and resets it to 3.</td>
</tr>
<tr>
<td><strong>Sort Entries</strong></td>
<td>When set to True, sorts results in ascending order. For details, see “Sort Order” on page 130.</td>
</tr>
</tbody>
</table>
**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 278.

### 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**

![Sample Static List Control](image)

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

![List Value Dialog Box](image)

Click *Add* to add list items. Each list item must have a unique key. The dialog box 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

| TextControl: |

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field width in pixels</td>
<td>Use this field to configure the field’s visible width on the form. The default is 200 pixels.</td>
</tr>
<tr>
<td>Lower bounds (for numbers only)</td>
<td>The lowest number allowed for decimal or integer values.</td>
</tr>
<tr>
<td>Maximum length</td>
<td>The maximum length for string values. Blocks input when this length is reached.</td>
</tr>
<tr>
<td>Minimum length</td>
<td>The minimum length for string values. Validates that the user enters a string at least this long.</td>
</tr>
<tr>
<td>Number of characters allowed</td>
<td>Specifies the number of characters a user is allowed to enter. This is related to Field width in pixels.</td>
</tr>
<tr>
<td>Upper bound (for numbers only)</td>
<td>The highest number allowed for decimal or integer values.</td>
</tr>
<tr>
<td>Validation Mask (regular expression)</td>
<td>An expression used for validating the field’s data. Designer provides a default set of validation masks by default. You must enable them through Windows &gt; Preferences &gt; Provisioning &gt; Validation Mask. For more information, see Section 2.3, “Setting Provisioning View Preferences,” on page 23.</td>
</tr>
</tbody>
</table>

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

| TextAreaControl: |

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of columns displayed</td>
<td>The visible width of the control; the number of characters wide.</td>
</tr>
<tr>
<td>Number of lines displayed</td>
<td>The number of lines to display at one time.</td>
</tr>
</tbody>
</table>
6.5.19 Title
Use this read-only control to label your form or provide instructions.

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

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

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

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

- DNDisplay
This section describes the specialized support, including the following:

- Section 6.6.1, “Formatting DNs,” on page 154.

### 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 controls 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 box (also called the object lookup dialog box) 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 155).

The object selector only supports attributes of String or DNLookup data types whose directory abstraction layer access properties for required and searchable are set to true.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity key used for object lookup</td>
<td>This is the key to the directory abstraction layer entity whose DN you want to search for or display. This is a required field.</td>
</tr>
<tr>
<td>Object selector type</td>
<td>paramlist: Causes the object selector dialog box to perform an object lookup. You specify the lookup criteria via the Entity key used for object lookup property.</td>
</tr>
<tr>
<td></td>
<td>container: Causes the object selector dialog box to display one or more containers for selection. The containers for searching are determined by the Search container property that is specified in the directory abstraction layer for the entity named in the Entity key used for object lookup property. For example, if the Entity key used for object lookup 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.</td>
</tr>
</tbody>
</table>
Creating Forms for a Provisioning Request Definition

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 resulting object selector is shown in Figure 6-32.
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 51.
   - 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 section 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 users 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

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

The `getCandidates` event is defined as a property on the `Candidates` Picklist control.
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 in Figure 6-34 on page 158). It passes the value of the `OfficeAreaCode` text field as the query’s input parameter `AreaCodeParam`.

**Figure 6-38  GetCandidates Event**
6.8 Printing Forms

You can add a Print button to a request or approval form using JavaScript. To add the ability to print a form, add the URL to the JavaScript library PrintForm.js. The library is located in the User Application WAR at this URL: ./javascript/JUICE/form/PrintForm.js. You can optionally do the following:

- Open the print form when doing a submit (after data is validated) by adding the following to the form onload event or one of the field’s onload events:

  ```javascript
  var action = "SubmitAction"
  var pf = new PrintForm(action);
  form.interceptAction("SubmitAction", "around",
                      pf.printFormInterceptor, pf);
  ```

![Figure 6-39 Sample Print Form with a Print Button](image)

To print on an approval action, the code would look like this:

```javascript
var action = "ApprovalAction";
var pf = new PrintForm(action);
form.interceptAction("ApprovalAction", "around",
                     pf.printFormInterceptor, pf);
```

- Add a Print button next to one of the fields on your form by adding this to the field’s onload event:

  ```javascript
  var ctrl = JUICE.UICtrlUtil.getControl(field.getName());
  var btn = JUICE.UICtrlUtil.addButton(ctrl, "printid", "Print",
                                      "Print", "javascript:var p = new PrintForm();
                                      p.printFormAfterValidation("printid");");
  ```
6.9 Providing Direct Access to a Form

If you want to users to be able to go directly to a form, you need to make some changes within the WAR file for the User Application and redeploy the WAR file. The configuration steps are described below.

1. In the WEB-INF/struts-config.xml file, specify roles="authUser" within the following action statement:

   ```xml
   <action path="/getAFResourceList" scope="request"
      name="apwaListForm"
      type="com.novell.srvprv.apwa.actions.GetResourceList"
      roles="authUser"/>
   ```

2. Also in the WEB-INF/struts-config.xml file, specify roles="guest, authUser" within the following action statement:

   ```xml
   <action path="/createAFResourceRequest" scope="request"
      name="apwaDetail"
      type="com.novell.srvprv.apwa.actions.ProcessResourceRequest"
      roles="guest,authUser"/>
   ```

3. Also in the WEB-INF/struts-config.xml file, add the following action mapping:

   ```xml
   <action path="/getGuestResourceList" scope="request"
      name="apwaListForm"
      type="com.novell.srvprv.apwa.actions.GetResourceList"
      roles="guest">
      <forward name="success" path="tiles.guestResourceList"/>
      <forward name="displaySelector" path="tiles.guestSelector"/>
   </action>
   ```

4. In the WEB-INF/tiles-defs.xml file, add the following definitions:

   ```xml
   <definition name="tiles.guestSelector" extends=".unauthenticatedLayout">
      <put name="title" value="JSP_TITLE_SELECTOR"/>
      <put name="body" value="/jsps/tiles/selector.jsp"/>
   </definition>
   <definition name="tiles.guestResourceList" extends=".unauthenticatedLayout">
      <put name="title" value="JSP_TITLE_RESOURCE_LIST"/>
      <put name="body" value="/jsps/tiles/resourceList.jsp"/>
   </definition>
   ```
5 Redeploy the WAR file.

The URL for accessing a form directly must use this format:
http://localhost:8080/IDMProv/
getAFResourceList.do?apwaLeftNavItem=JSP_MENURESOURCE_REQUEST_CLASSIC
&wfdn=PRDEFDN

The parameter PRDEFDN must be replaced with the DN of the request definition. Here’s a complete example that includes this parameter:
http://localhost:8080/IDMProv/
getAFResourceList.do?apwaLeftNavItem=JSP_MENURESOURCE_REQUEST_CLASSIC
&wfdn=cn=TestPRDEF,cn=RequestDefs,cn=AppConfig,cn=Test-Antelope,cn=TestDrivers,o=novell
Creating the Workflow for a Provisioning Request Definition

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

- Section 7.1, “About the Workflow Tab,” on page 165
- Section 7.2, “Adding Activities to a Workflow,” on page 170
- Section 7.3, “Adding Flow Paths,” on page 173
- Section 7.4, “Configuring Flow Paths,” on page 174
- Section 7.5, “Addressing an Approval Activity,” on page 176
- Section 7.6, “Provisioning Multiple Individuals with One Workflow Instance,” on page 181
- Section 7.7, “Working with Entity Activities,” on page 183
- Section 7.8, “Making Distinguished Name References Portable,” on page 185
- Section 7.9, “Configuring Digital Signature Support,” on page 185

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.
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 activity and Finish activity.

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

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>Deletes the selected activity or flow path.</td>
</tr>
</tbody>
</table>
Creating the Workflow for a Provisioning Request Definition

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**

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Show Activity IDs</strong></td>
<td>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 Window &gt; Preferences &gt; Provisioning &gt; Workflows &gt; Diagram Preferences &gt; Show Activity IDs.</td>
</tr>
<tr>
<td><strong>Show Flow Path Types</strong></td>
<td>Turns the display of flow path types (for example, forward, approved, denied) on and off. When Show Flow Path Types is turned on, a label is displayed on each flow path indicating the flow path type.</td>
</tr>
<tr>
<td><strong>Show Properties</strong></td>
<td>Displays the Properties view for the selected activity. If no activity is selected, displays the Properties view for the Workflow itself.</td>
</tr>
<tr>
<td><strong>Show Data Item Mapping</strong></td>
<td>Displays the Data Item Mapping view for the selected activity.</td>
</tr>
<tr>
<td><strong>Show E-Mail Notification</strong></td>
<td>Displays the E-Mail Notification view for the selected activity.</td>
</tr>
</tbody>
</table>

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:
The palette includes the following tools:

**Table 7-2  Workflow Palette**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>Selects individual nodes or flow paths. To select a node, click the Select tool, then click a node.</td>
</tr>
<tr>
<td>Marquee</td>
<td>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, &quot;Views,&quot; on page 169 for information about Views).</td>
</tr>
<tr>
<td>Flow Path</td>
<td>Creates flow paths between nodes. Flow paths provide connection logic for connecting nodes. For information about connecting nodes, see Section 7.3, &quot;Adding Flow Paths,&quot; on page 173.</td>
</tr>
<tr>
<td>Activities (for example, Start, Approval, Log)</td>
<td>Inserts the selected activity into the workflow. For information about adding activities, see Section 7.2, &quot;Adding Activities to a Workflow,&quot; on page 170. For detailed descriptions of the activities, see Chapter 8, &quot;Workflow Activity Reference,&quot; on page 189.</td>
</tr>
</tbody>
</table>
7.1.3 Views

The Workflow page also includes the Properties, Data Item Mapping, and E-Mail 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**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Properties</th>
<th>E-Mail Notification</th>
<th>Data Item Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Approval</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Log</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branch</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Merge</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mapping</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Workflow Status</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>E-Mail</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Finish</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Integration</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Entitlement</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Entity</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
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 activity 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 Flow Paths,” on page 173. 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 170). For detailed information about configuring the different types of activities, see Chapter 8, “Workflow Activity Reference,” on page 189 and Chapter 9, “Working with Integration Activities,” on page 219.

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 189.
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**

<table>
<thead>
<tr>
<th>Date</th>
<th>Activity</th>
<th>User</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/14/2019</td>
<td>First</td>
<td>Admin</td>
<td>User task assigned to reviewer John Doe</td>
</tr>
<tr>
<td>11/14/2019</td>
<td>Second</td>
<td>Admin</td>
<td>User task assigned to reviewer Jane Smith</td>
</tr>
</tbody>
</table>

**Figure 7-6 Activities in User Comments Using Descriptive Names**

<table>
<thead>
<tr>
<th>Date</th>
<th>Activity</th>
<th>User</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/14/2019</td>
<td>First</td>
<td>Admin</td>
<td>User task assigned to reviewer John Doe</td>
</tr>
<tr>
<td>11/14/2019</td>
<td>Second</td>
<td>Admin</td>
<td>User task assigned to reviewer Jane Smith</td>
</tr>
</tbody>
</table>

### 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:
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 189.

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 129 for information about setting the properties of form fields.

7.2.3 Defining the E-Mail Notification Settings

You use the E-Mail 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 E-Mail Notification from the menu.

TIP: You can also display the E-Mail Notification tab by selecting Show E-Mail Notification from the PRD menu.
The E-Mail Notification view is displayed:

2 Click the E-Mail 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.

   Localized e-mail templates By default, Designer displays the default e-mail notification templates. When you select a default template, the e-mail is in the user’s default language (if the default is a supported language). You can set the Show all localized e-mail templates preference to True so that Designer also allows you to select from the list of localized e-mail templates. The localized templates have the same name as the default, but the Java language code is appended to the name of the e-mail template. For example, cn=Provisioning Notification Activity_es, cn=Default Notification Collection, cn=security indicates this is the Spanish language version of this template. When you select a localized template, the e-mail is in the language of the template regardless of the user’s default language.

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 189 for information about e-mail notification settings.

7.3 Adding 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:

![Diagram of workflow with Start, Approve, and Finish activities connected by flow paths.]

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 174.

### 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 Flow Paths,” on page 173.
To configure a flow path:

1. Click the flow path in the workflow diagram.

2. 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:
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>
<thead>
<tr>
<th>Type of Expression</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual user DN</td>
<td>'cn=jdoe,ou=users,ou=mysample,o=myorg'</td>
</tr>
<tr>
<td>Individual group DN</td>
<td>'cn=Accounting,ou=groups,ou=mysample,o=myorg'</td>
</tr>
<tr>
<td>A vector of DNs (can include user or group DNs)</td>
<td>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();</td>
</tr>
</tbody>
</table>

In this example, the total number of addressees is four (three individuals and one user from the Accounting group).
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*

<table>
<thead>
<tr>
<th>Addressee Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual User DN or Entity</td>
<td>✷ Only the user can see the Approval activity in their task list.</td>
</tr>
<tr>
<td></td>
<td>✷ Only one approval is needed to complete the activity as Approved.</td>
</tr>
<tr>
<td>Individual Group DN or Group Entity</td>
<td>✷ Each member of Group can see the activity in the task list.</td>
</tr>
<tr>
<td></td>
<td>✷ When one member claims the activity, it is removed from the task lists of others.</td>
</tr>
<tr>
<td></td>
<td>✷ Only one approval is needed to complete the activity as Approved.</td>
</tr>
<tr>
<td>Multiple User DNs or User Entities (Virtual Group of Users)</td>
<td>Not allowed.</td>
</tr>
<tr>
<td>Multiple Group DNs or Group Entities (Virtual Group of Groups)</td>
<td>Not allowed.</td>
</tr>
<tr>
<td>Mixture of Users and Groups (Virtual Group Mixture)</td>
<td>Not allowed.</td>
</tr>
</tbody>
</table>

**Group DNs and Proxy Processing**

If a workflow is assigned to a Group and e-mail notification is used for the approvals, all members of the group are sent an e-mail. If a proxy user is assigned to any members of the group, the processing works as follows:

- If the approver is a single user then the e-mail notification is sent to both users (the original and proxy users).
- If the approver is a group DN and one of the users in the group is assigned a proxy user, the user who is the proxy is NOT notified by e-mail when a new request is placed in the task list.

If you want the proxy user to be notified by e-mail, assign the approval task to the members of the group and set the approver type to *Group Approver*. For example, if you assign the approval activity to:

```
IDVault.get('cn=Marketing,ou=groups,ou=idmsample,o=novell', 'group', 'Member')
```
When you set the approval type to Group, a notification is sent to each member's proxy, if the member has a proxy. One member of the group can claim and act on the approval task which is the same behavior as if you assigned it directly to the group DN.

**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*

<table>
<thead>
<tr>
<th>Addressee Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual User DN or Entity</td>
<td>• Only the user can see the Approval activity in their task list.</td>
</tr>
<tr>
<td></td>
<td>• Only one approval is needed to complete the activity as Approved.</td>
</tr>
<tr>
<td>Individual Group DN or Group Entity</td>
<td>• Each member of Group can see the activity in their task list.</td>
</tr>
<tr>
<td></td>
<td>• When one member claims the activity, it is removed from task lists of others.</td>
</tr>
<tr>
<td></td>
<td>• Only one approval is needed to complete the activity as Approved.</td>
</tr>
<tr>
<td>Multiple User DNs or User Entities (Virtual Group of Users)</td>
<td>• Each user in the virtual group can see the activity in their task list.</td>
</tr>
<tr>
<td></td>
<td>• When one user from the virtual group claims the activity, the activity is removed from the task lists of others.</td>
</tr>
<tr>
<td></td>
<td>• Only one approval is needed to complete the activity as Approved.</td>
</tr>
<tr>
<td>Multiple Group DNs or Group Entities (Virtual Group of Groups)</td>
<td>• Each member in each of the groups can see the activity in their task list.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• Only one approval is needed to complete the activity as Approved.</td>
</tr>
<tr>
<td>Mixture of Users and Groups (Virtual Group Mixture)</td>
<td>• Each user and member of each Group of the mixed virtual group can see the activity in their task list.</td>
</tr>
<tr>
<td></td>
<td>• When one member from the virtual group claims the activity, the activity is removed from the task lists of others.</td>
</tr>
<tr>
<td></td>
<td>• Only one approval is needed to complete the activity as Approved.</td>
</tr>
</tbody>
</table>

**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*

<table>
<thead>
<tr>
<th>Addressee Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual User DN or Entity</td>
<td>• Only the user can see the activity in their task list.</td>
</tr>
<tr>
<td></td>
<td>• Only one approval is needed to complete the activity as Approved.</td>
</tr>
</tbody>
</table>
### 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**

<table>
<thead>
<tr>
<th>Addressee Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual User DN or Entity</td>
<td>• Only the user can see the activity in their task list.</td>
</tr>
<tr>
<td></td>
<td>• Only one approval is needed to complete the activity as Approved.</td>
</tr>
<tr>
<td>Individual Group DN or Group Entity</td>
<td>• Each member of the group can see the activity in their task list.</td>
</tr>
<tr>
<td></td>
<td>• When one member claims the activity, the activity is removed from the</td>
</tr>
<tr>
<td></td>
<td>task lists of others.</td>
</tr>
<tr>
<td></td>
<td>• Only one approval is needed to complete the activity as Approved.</td>
</tr>
<tr>
<td>Multiple User DNs or User Entities (Virtual Group of</td>
<td>• Each user in the virtual group can see the activity in their task list.</td>
</tr>
<tr>
<td>Users)</td>
<td>• Each user can claim the activity.</td>
</tr>
<tr>
<td></td>
<td>• Approval of each user is needed to complete the activity as Approved.</td>
</tr>
<tr>
<td></td>
<td>• Any single denial completes the activity as Denied.</td>
</tr>
<tr>
<td>Multiple Group DNs or Group Entities (Virtual Group</td>
<td>• Each member in each of the groups can see the activity in their task list.</td>
</tr>
<tr>
<td>of Groups)</td>
<td>• When one member from a group claims the activity, the activity is</td>
</tr>
<tr>
<td></td>
<td>removed from the task list of others in that Group.</td>
</tr>
<tr>
<td></td>
<td>• Each group must supply one approval to complete the activity as Approved.</td>
</tr>
<tr>
<td></td>
<td>• Any single denial completes the activity as Denied.</td>
</tr>
<tr>
<td>Mixture of Users and Groups (Virtual Group Mixture)</td>
<td>• Each user and each member of each group of the mixed virtual group can</td>
</tr>
<tr>
<td></td>
<td>see the activity in their task list.</td>
</tr>
<tr>
<td></td>
<td>• Each user can claim the activity, and one member of each group can claim</td>
</tr>
<tr>
<td></td>
<td>the activity (then others in the group do not see the task.)</td>
</tr>
<tr>
<td></td>
<td>• Each user and one member of each group must approve to complete the</td>
</tr>
<tr>
<td></td>
<td>activity as Approved.</td>
</tr>
<tr>
<td></td>
<td>• Any single denial completes the activity as Denied.</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Addressee Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual Group DN or Group Entity</td>
<td>• Each member of the group can see the activity in their task list.</td>
</tr>
<tr>
<td></td>
<td>• When one member claims the activity, the activity is removed from the</td>
</tr>
<tr>
<td></td>
<td>task lists of others.</td>
</tr>
<tr>
<td></td>
<td>• Only one approval is needed to complete the activity as Approved.</td>
</tr>
<tr>
<td>Multiple User DNs or User Entities (Virtual Group of</td>
<td>• Each user in the virtual group can see the activity in their task list.</td>
</tr>
<tr>
<td>Users)</td>
<td>• All users in the virtual group can claim the activity simultaneously.</td>
</tr>
<tr>
<td></td>
<td>• An absolute number or specified percentage of Addressees must approve to complete the activity as Approved.</td>
</tr>
</tbody>
</table>
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 panel, 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.

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

5 Click the name of the workflow.

A form is displayed that provides three 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 workflow that uses the Single Flow Provision Members flow strategy, you need to perform these additional setup steps.

1. Log in to iManager as an administrator.
2. In Roles and Tasks, select Provisioning Configuration.
3. Select Provisioning Teams.
4. Set up the team if it is not already set up.
5. Bind the workflow to the team by defining a Provisioning Team Request using the Provisioning Configuration Role and Task.

### 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 183
- Section 7.7.2, “Using an Entity Activity to Delete an Entity,” on page 184
- Section 7.7.3, “Using an Entity Activity to Delete an Attribute or Value,” on page 184

#### 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 45 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 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 Making Distinguished Name References Portable

When you use a DN in an expression in a provisioning request definition, the expression might fail if you deploy the provisioning request definition to an Identity Vault with a different structure. You typically specify DNs in:

- Overview panel: Trustee specification.
- User activity: Addressee and escalation addressee.
- Entity activity: Entitlement reference and entity DN.
- Many other expressions, for example, IDVault.get(dn, class, attribute).

Some expressions, such as recipient, are portable. The following expressions, which are used by default in the User activity, are also portable:

```javascript
IDVault.get(recipient,'user','manager')
IDVault.get(approval_A.getAddressee(),'user','manager')
```

To ensure that your DN expressions are portable across Identity Vaults, you can use one of the following variables:

- ROOT_CONTAINER: for example ou=idm-prov,o=novell
- PROVISIONING_DRIVER: for example cn=UserApplication,cn=TestDrivers,o=novell
- USER_CONTAINER: for example ou=users,ou=idm-prov,o=novell
- GROUP_CONTAINER: for example ou=groups,ou=idm-prov,o=novell

These variables are defined during installation of the user application and are resolved at runtime by the ECMAScript engine. You can find them in the ECMA expression builder under the process node. Suppose you wanted to reference an entitlement at the following DN:

'cn=myEntitlement,cn=UserApplication,cn=TestDrivers,o=novell'

You could use the following expression to make the DN portable to any identity vault:

```
'cn=MyEntitlement,' + PROVISIONING_DRIVER
```

You can use this technique for users and groups also.

**NOTE:** Trustees are not expressions so you cannot use this technique with Trustees.

7.9 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>
<thead>
<tr>
<th>Step</th>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create one or more digital signature declarations.</td>
<td>See Section 7.9.2, “Creating a Signature Declaration,” on page 187.</td>
</tr>
</tbody>
</table>
Table 7-10  Digital Signature Workflow Properties

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Signature Type</td>
<td>Specifies whether the digital signature uses data or form as its type:</td>
</tr>
<tr>
<td></td>
<td>- <strong>Data</strong>: Specifies that the XML signature serves as the user agreement. When you select <em>Data</em>, the XML data is written to the audit log.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Form</strong>: Specifies to generate a PDF document that includes the digital signature declaration. This document serves as the user agreement. The user can preview the generated PDF document before submitting a request or approval. When you select <em>Form</em>, the PDF document (encapsulated in XML) is written to the audit log.</td>
</tr>
<tr>
<td>Digital Signature Declaration</td>
<td>Specifies a digital signature confirmation string that certifies the user’s signature. See Section 7.9.2, “Creating a Signature Declaration,” on page 187.</td>
</tr>
</tbody>
</table>
7.9.2 Creating a Signature Declaration

1 Open the Signature Declarations tab.

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature Declaration ID</td>
<td>A unique identifier for the signature declaration. This ID is displayed in the drop-down for the “Digital Signature Declaration” on page 186.</td>
</tr>
<tr>
<td>Language</td>
<td>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 Export &gt; Export Localization Data so that you can send the declaration to be localized as part of the rest of the User Application display labels and strings.</td>
</tr>
<tr>
<td>Signature Declaration</td>
<td>The string to display in a form as the signature declaration.</td>
</tr>
</tbody>
</table>

3 Click Save.
This section provides details on configuring the different types of workflow activities. Topics in this section include:

- Section 8.1, “Start Activity,” on page 189
- Section 8.2, “Approval Activity,” on page 191
- Section 8.3, “Log Activity,” on page 200
- Section 8.4, “Branch Activity,” on page 202
- Section 8.5, “Merge Activity,” on page 202
- Section 8.6, “Condition Activity,” on page 203
- Section 8.7, “Mapping Activity,” on page 204
- Section 8.8, “Workflow Status,” on page 205
- Section 8.9, “E-Mail Activity,” on page 206
- Section 8.10, “Finish Activity,” on page 208
- Section 8.11, “Integration Activity,” on page 210
- Section 8.12, “Entitlement Activity,” on page 213
- Section 8.13, “Entity Activity,” on page 214

The display names for all activities can be localized by clicking the *Localize Strings* button (see Section 2.10, “Localizing Display Labels,” on page 36) for the activity name property. Activity display names are also exported as part of the Provisioning view’s *Export > Export Localization Data* (see Section 2.10.3, “Exporting and Importing Localized Labels,” on page 37) so that you can send the activity names to be localized as part of the rest of the User Application display labels and strings.

### 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 109.

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.

- Section 8.1.1, “Properties,” on page 190
- Section 8.1.2, “Data Item Mapping,” on page 191
8.1.1 Properties

The Start activity has the following property:

Table 8-1  Start Activity Property

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Provides a name for the activity.</td>
</tr>
<tr>
<td>Display Name Override</td>
<td>Allows you to override the provisioning request definition’s display name. This is the name that displays in the Resource column when the user selects My Requests or Team Requests. The name can be a constant or the result of an ECMA expression, and it can be localized for each supported locale. To access the dialog boxes that let you specify the constant or ECMA expression, the Localize Strings button in the Value column.</td>
</tr>
</tbody>
</table>

To specify a constant, click and type the value in the field. The value is displayed exactly as entered in this field.

To specify an ECMA expression, click and specify the expression.

If you specify both a constant and ECMA expression, the runtime displays whichever value was the last one entered as shown in the dialog box (above).
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

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

For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 267.

8.1.3 E-Mail 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 109.

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.

- Section 8.2.1, “Properties,” on page 192
- Section 8.2.2, “Data Item Mapping,” on page 198
- Section 8.2.3, “E-Mail Notification,” on page 198

### 8.2.1 Properties

The Approval activity has the following properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Provides a name for the activity.</td>
</tr>
<tr>
<td>Addressee</td>
<td>Specifies a dynamic expression that identifies the addressee for the activity. The addressee is determined at runtime, based on how the expression is evaluated.</td>
</tr>
</tbody>
</table>

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 176.

**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.

For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 267. For descriptions of the system variables available in a workflow, see Section 4.3.3, “Understanding Workflow Data,” on page 90.
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reminder Start</strong></td>
<td>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 predefined expressions that represent common intervals (for example, hour, day, week) in the ECMA\texttt{Script Variables} pane of the ECMA expression builder.</td>
</tr>
<tr>
<td></td>
<td>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 <strong>Reminder Start</strong> time, a <strong>Reminder Interval</strong>, and the e-mail to be sent (see Section 8.2.3, “E-Mail Notification,” on page 198). For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 267. For descriptions of the system variables available in a workflow, see Section 4.3.3, “Understanding Workflow Data,” on page 90.</td>
</tr>
<tr>
<td><strong>Reminder Interval</strong></td>
<td>Specifies a dynamic expression that defines the interval between which reminder e-mails are sent. You can pick predefined expressions that represent common intervals (for example, hour, day, week) in the ECMA\texttt{Script Variables} pane of the ECMA expression builder.</td>
</tr>
<tr>
<td><strong>Escalation Addressee</strong></td>
<td>Not available when the approver type is <em>Multiple</em> or <em>Quorum</em></td>
</tr>
<tr>
<td></td>
<td>Specifies a dynamic expression that identifies the user who should get this task if the timeout limit has been reached.</td>
</tr>
<tr>
<td></td>
<td>The escalation addressee is determined at runtime, based on how the expression is evaluated.</td>
</tr>
<tr>
<td></td>
<td>For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 267. For descriptions of the system variables available in a workflow, see Section 4.3.3, “Understanding Workflow Data,” on page 90.</td>
</tr>
</tbody>
</table>


### Escalation Count

Not available when the approver type is *Multiple* or *Quorum*.

Specifies the number of times to retry the activity in the event of a timeout.

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.

The *Timeout* interval (see *Timeout* in this table) takes precedence over the *Escalation Interval*. 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 finishes 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.

### Escalation Interval

Not available when approver type is *Multiple* or *Quorum*.

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.

The *Timeout* interval (see *Timeout* in this table) takes precedence over the *Escalation Interval*. 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.

For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 267. For descriptions of the system variables available in a workflow, see Section 4.3.3, “Understanding Workflow Data,” on page 90.

### Escalation Reminder Start

Not available when the approver type is *Multiple* or *Quorum*.

Specifies a dynamic expression that defines the time at which the first reminder e-mail (see *Reminder Start* in this table) should be sent to the *Escalation Addressee*. The start value is an offset from the time of the escalation assignment. You can pick predefined expressions that represent common intervals (for example, hour, day, week) in the *ECMAScript Variables* pane of the ECMA expression builder.
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Escalation Reminder Interval</td>
<td>Not available when the approver type is Multiple or Quorum. Specifies a dynamic expression that defines how often messages are sent to the Escalation Addresssee after the first escalation reminder is sent. You can pick predefined expressions that represent common intervals (for example, hour, day, week) in the ECMAScript Variables pane of the ECMA expression builder.</td>
</tr>
<tr>
<td>Final Timeout Action</td>
<td>Determines the final state of the request in the event that the workflow times out. The choices are <strong>approved</strong>, <strong>denied</strong>, <strong>refused</strong>, <strong>timedout</strong>, <strong>error</strong></td>
</tr>
<tr>
<td>Timeout</td>
<td>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. The Timeout setting takes precedence over the Escalation Count and Escalation Interval 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 finishes 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.</td>
</tr>
<tr>
<td>Time Units</td>
<td>Determines the unit of measure used for the timeout interval. The choices are <strong>Milliseconds</strong>, <strong>Days</strong>, <strong>Hours</strong>, <strong>Minutes</strong>, <strong>Seconds</strong></td>
</tr>
</tbody>
</table>

For details on building ECMA expressions, see Chapter 10, "Working with ECMA Expressions," on page 267. For descriptions of the system variables available in a workflow, see Section 4.3.3, "Understanding Workflow Data," on page 90.
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>Specifies the name of the approval form to display at runtime, or lets you define a new form. Select the name of the form you want to use or create new form. When you choose to create a new form, the Create New Form Wizard launches and looks similar to this. Select the data items to include in the form from the data items listed, then click Finish. The Approval Form Wizard generates each of the selected data items as a String type field in the new form.</td>
</tr>
<tr>
<td>Exclude Requestor</td>
<td>Specifies whether requestors can approve their own provisioning requests. Select True to allow a requestor to approve their own provisioning requests; otherwise, select False.</td>
</tr>
</tbody>
</table>

Select the data items to include in the form from the data items listed, then click Finish. The Approval Form Wizard generates each of the selected data items as a String type field in the new form.

An Approval activity must have a form associated with it. If no form is specified, an error message is displayed at runtime.
### Property Name | Description
--- | ---
**Approver Type** | Specifies the number of addresses that are allowed and the approval pattern that is enforced for this activity. The choices are:
- **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.

You cannot use post activity data item mapping with the **Multiple Approver Type**.

- **Quorum**: Action by a percentage of addressees or an absolute number of addressees (see Quorum property in this table) is required to complete the approval.

You cannot use post-activity data item mapping with the **Quorum Approver Type**.

For information about how the **Approver Type** property interacts with the **Addressssee** property, see Section 7.5, “Addressing an Approval Activity,” on page 176.

**Notify by E-Mail** | Specifies whether this activity should send e-mail notifications. Set to True to notify by e-mail; otherwise, set to False.

You specify the e-mail to send using the E-Mail Notification tab (see Section 8.2.3, “E-Mail Notification,” on page 198).

To use this feature, the **Notify participants by E-Mail** parameter for the provisioning request definition must be set to True (see Table 5-1, “Overview Properties,” on page 105).

**Quorum** | Not available when approver type is Normal, Group, or Multiple.

Allows you to specify a constant value or to create an ECMA 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.

**Digital Signature Type** | See Digital Signature Type in Table 7-10 on page 186.

**Priority** | 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, `flowdata.get("Priority")`.

In the User Application, users can sort the list of tasks by the priority values of the tasks.

**NOTE**: To enable delegation to a group DN, you can have an approver type of Group or Normal, but the Addressssee value must be an expression that returns the user DN for each member of that group. For example, `IDVault.get(groupdn, ‘sales’, ‘members’)`
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>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre Activity</td>
<td>Allows you to specify one or more pre-activity mappings. When this option is selected, you can double-click a cell in the Source Expression column to specify where the approval form gets data for a particular target form field.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> When the Pre-Activity choice is selected, the cells in the Target Form Field column are not editable.</td>
</tr>
<tr>
<td>Post Activity</td>
<td>Allows you to specify one or more Post Activity mappings. When this option is selected, you can double-click a cell in the Target Expression column to specify where data from a form field should be copied after the form has been processed.</td>
</tr>
<tr>
<td></td>
<td>You cannot use Post Activity mapping with the Multiple and Quorum approver types (see Section 8.2.1, “Properties,” on page 192).</td>
</tr>
<tr>
<td></td>
<td>The form for an Approval activity includes a special internal control called apwaComment. 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 139.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> When the Post-Activity option is selected, the cells in the Source Form Field column are not editable.</td>
</tr>
<tr>
<td>Source Expression</td>
<td>Specifies a source expression for a pre-activity mapping. When you click a cell in the Source Expression column, the ECMA expression builder displays to help you define your expression.</td>
</tr>
<tr>
<td>Target Expression</td>
<td>Specifies a target expression for a post-activity mapping. When you click a cell in the Target Expression column, the ECMA expression builder displays to help you define your expression.</td>
</tr>
</tbody>
</table>

For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 267.

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

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notify</td>
<td>Specifies that this e-mail notification is a notification e-mail.</td>
</tr>
<tr>
<td>Reminder</td>
<td>Specifies that this e-mail notification is a reminder e-mail.</td>
</tr>
<tr>
<td>Retry Reminder</td>
<td>Specifies that this e-mail notification is a retry reminder e-mail.</td>
</tr>
<tr>
<td>Show System Tokens</td>
<td>Displays system tokens (for example, TO, CC, BCC, REPLYTO) in the Target column.</td>
</tr>
<tr>
<td>E-Mail Template</td>
<td>Specifies the name of the e-mail template to use. By default, the Approval activity uses the Provisioning Notification template. You can edit an e-mail template in Designer. For more information, see &quot;Editing an e-mail template:&quot; on page 173.</td>
</tr>
</tbody>
</table>
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.

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.

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.

- Section 8.3.1, “Properties,” on page 201
- Section 8.3.2, “Data Item Mapping,” on page 201
- Section 8.3.3, “E-Mail Notification,” on page 201

8.3.1 Properties

The Log activity has the following properties:

Table 8-6  Log Activity Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Provides a name for the activity.</td>
</tr>
<tr>
<td>Audit</td>
<td>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.</td>
</tr>
<tr>
<td>Author</td>
<td>Defines the author for the message. By default, the author is the initiator of the provisioning request.</td>
</tr>
<tr>
<td>Message</td>
<td>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. For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 267. For descriptions of the system variables available in a workflow, see Section 4.3.3, “Understanding Workflow Data,” on page 90.</td>
</tr>
<tr>
<td>Comment</td>
<td>Specifies an ECMA expression that defines text that can be displayed in the user comments. You might use it to record the reason for a request or a request’s completed approval status. Some examples include: &quot;Reason for request: &quot; + flowdata.get(‘reason’) or &quot;Process has been &quot; + flowdata.get(IDM_COMPLETED_APPROVAL_STATUS’)</td>
</tr>
</tbody>
</table>

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.

- Section 8.4.1, “Properties,” on page 202
- Section 8.4.2, “Data Item Mapping,” on page 202
- Section 8.4.3, “E-Mail Notification,” on page 202

8.4.1 Properties

The Branch activity has the following property:

Table 8-7  Branch Activity Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Provides a name for the activity.</td>
</tr>
</tbody>
</table>

8.4.2 Data Item Mapping

Not supported with this activity.

8.4.3 E-Mail Notification

Not supported with this activity.

8.5 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.

- Section 8.5.1, “Properties,” on page 203
- Section 8.5.2, “Data Item Mapping,” on page 203
- Section 8.5.3, “E-Mail Notification,” on page 203
8.5.1 Properties

The Merge activity has the following property:

Table 8-8  Merge Activity Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Provides a name for the activity.</td>
</tr>
</tbody>
</table>

8.5.2 Data Item Mapping

Not supported with this activity.

8.5.3 E-Mail Notification

Not supported with this activity.

8.6 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.

- Section 8.6.1, “Properties,” on page 203
- Section 8.6.2, “Data Item Mapping,” on page 204
- Section 8.6.3, “E-Mail Notification,” on page 204

8.6.1 Properties

The Condition activity has the following properties:

Table 8-9  Condition Activity Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Provides a name for the activity.</td>
</tr>
</tbody>
</table>
8.6.2 Data Item Mapping

Not supported with this activity.

8.6.3 E-Mail Notification

Not supported with this activity.

8.7 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.

8.7.1 Properties

The Mapping activity has the following property:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition Expression</td>
<td>Specifies an ECMA expression that returns True or False. The value returned determines which flow path is followed after the activity has finished executing.</td>
</tr>
<tr>
<td>TIP:</td>
<td>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: (approval_A.getAction() == &quot;DENIED&quot;) instead of this one: (approval_A.getAction()).equals(&quot;DENIED&quot;)</td>
</tr>
</tbody>
</table>
8.7.2 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 8-11  Mapping Activity Data Item Mappings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Expression</td>
<td>Specifies a source expression. When you click a cell in the Source Expression column, the ECMA expression builder displays to help you define your expression. For example,</td>
</tr>
<tr>
<td></td>
<td>function list() { s=new java.lang.String();</td>
</tr>
<tr>
<td></td>
<td>if (wi.XPath('count(flow-data/groups)') &gt; 0) s=&quot;There was a group selected&quot;; return s;}; list();</td>
</tr>
<tr>
<td>Target Expression</td>
<td>Specifies a target expression. When you click a cell in the Target Expression 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:</td>
</tr>
<tr>
<td></td>
<td>flowdata.testexpression</td>
</tr>
</tbody>
</table>

8.7.3 E-mail 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).

- Section 8.8.1, “Properties,” on page 205
- Section 8.8.2, “Data Item Mapping,” on page 206
- Section 8.8.3, “E-Mail Notification,” on page 206

8.8.1 Properties

The Workflow Status activity has the following properties:
8.8.2 Data Item Mapping

Not supported with this activity.

8.8.3 E-Mail Notification

Not supported with this activity.

8.9 E-Mail Activity

The E-Mail activity provides a way to send an e-mail to interested parties outside of an approval process.

- Section 8.9.1, “Properties,” on page 206
- Section 8.9.2, “Data Item Mapping,” on page 207
- Section 8.9.3, “E-Mail Notification,” on page 207

8.9.1 Properties

The E-Mail activity has the following properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Provides a name for the activity.</td>
</tr>
<tr>
<td>Notify by E-Mail</td>
<td>Specifies whether this activity should send e-mail notifications. Set to True to notify by e-mail; otherwise, set to False.</td>
</tr>
</tbody>
</table>

You specify the e-mail to send using the E-Mail Notification tab (see Section 8.9.3, “E-Mail Notification,” on page 207).

To use this feature, the Notify participants by E-Mail parameter for the provisioning request definition must be set to true (see Table 5-1, “Overview Properties,” on page 105).
8.9.2 Data Item Mapping

Not supported with this activity.

8.9.3 E-Mail Notification

To enable e-mail notification for this 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  E-mail Notification Settings for the E-Mail Activity

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-Mail Template</td>
<td>Specifies the name of the e-mail template to use. By default, the Approval</td>
</tr>
<tr>
<td></td>
<td>activity uses the Provisioning Notification template.</td>
</tr>
<tr>
<td></td>
<td>You can edit an e-mail template in Designer. For more information, see “</td>
</tr>
<tr>
<td></td>
<td>Editing an e-mail template:” on page 173.</td>
</tr>
</tbody>
</table>
NOTE: E-mail notification is only supported when the Notify participants by E-Mail check box is selected on the Overview tab, and the Notify by E-Mail property for the activity is set to True.

### 8.10 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.

- Section 8.10.1, “Properties,” on page 209
- Section 8.10.2, “Data Item Mapping,” on page 209
- Section 8.10.3, “E-mail Notification,” on page 209
8.10.1 Properties

The Finish activity has the following properties:

Table 8-15  Finish Activity Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Provides a name for the activity.</td>
</tr>
<tr>
<td>Notify by E-Mail</td>
<td>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. See Section 8.10.3, “E-mail Notification,” on page 209 for information about setting up the e-mail notification.</td>
</tr>
</tbody>
</table>

8.10.2 Data Item Mapping

Not supported with this activity.

8.10.3 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 8-16  E-Mail Notification Settings for the Finish Activity

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-Mail Template</td>
<td>Specifies the name of the e-mail template to use. By default, the Finish activity uses the Provisioning Approval Completed Notification template. You can edit an e-mail template in Designer. See “Editing an e-mail template:” on page 173 for more information.</td>
</tr>
</tbody>
</table>
### 8.11 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 219.

- Section 8.11.1, “Properties,” on page 211
- Section 8.11.2, “Data Item Mapping,” on page 211

---

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source</strong></td>
<td>Specifies the source expressions for target tokens in the e-mail body.</td>
</tr>
</tbody>
</table>
| **Target** | 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.

The available target tokens for the Provisioning Approval Completed Notification e-mail template are listed below:

- TO
- CC
- BCC
- REPLYTO
- requestStatus
- requestSubmissionTime
- requestID
- recipientFullName
- initiatorFullName
- requestTitle

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.

**NOTE:** When you create a workflow for use with the Resource Request portlet, and you use _default_ as the expression for the TO token, the addressee expression must be an IDVault expression.

For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 267.

---

**NOTE:** E-mail notification is only supported when the Notify participants by E-Mail check box is selected on the Overview tab.
Section 8.11.3, “E-Mail Notification,” on page 212

8.11.1 Properties

The Integration activity has the following properties.

Table 8-17  Integration Activity Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Provides a name for the activity.</td>
</tr>
<tr>
<td>WSDL Resource</td>
<td>Specifies a WSDL file for the Web service to be used in the Integration activity. After it is specified, the WSDL is incorporated into the provisioning request definition file. 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.</td>
</tr>
<tr>
<td>Timeout</td>
<td>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. For details on building ECMA expressions, see Chapter 10, “Working with ECMA Expressions,” on page 267. For descriptions of the system variables available in a workflow, see Section 4.3.3, “Understanding Workflow Data,” on page 90.</td>
</tr>
<tr>
<td>Retry Count</td>
<td>Specifies the number of times to retry the activity in the event of a timeout. 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.</td>
</tr>
<tr>
<td>Final Timeout Action</td>
<td>Determines the final state of the request in the event that the Integration activity times out. The choices are • success • fault • error • timedout</td>
</tr>
</tbody>
</table>

8.11.2 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 221.

Table 8-18 Integration Activity Data Item Mappings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Activity</td>
<td>Allows you to specify one or more pre-activity mappings.</td>
</tr>
<tr>
<td></td>
<td>When this option is selected, you can double-click a cell in the</td>
</tr>
<tr>
<td></td>
<td>Source Expression column to specify where the Integration activity gets</td>
</tr>
<tr>
<td></td>
<td>data for a particular Web service input field.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> When the Pre-Activity option is selected, the cells in the</td>
</tr>
<tr>
<td></td>
<td>Web Service Input Field column are not editable.</td>
</tr>
<tr>
<td>Post-Activity</td>
<td>Allows you to specify one or more post-activity mappings.</td>
</tr>
<tr>
<td></td>
<td>When this radio button is selected, you can double-click a cell in the</td>
</tr>
<tr>
<td></td>
<td>Target Expression column to specify where data from a Web service output</td>
</tr>
<tr>
<td></td>
<td>field should be copied after the form has been processed.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> When the Post-Activity option is selected, the cells in the</td>
</tr>
<tr>
<td></td>
<td>Web Service Output Field column are not editable.</td>
</tr>
<tr>
<td>Source Expression</td>
<td>Specifies a source expression. When you click a cell in the Source</td>
</tr>
<tr>
<td></td>
<td>Expression column, the ECMA expression builder displays to help you define</td>
</tr>
<tr>
<td></td>
<td>your expression. For example, flowdata.get('Start/RequestRate/Country1')</td>
</tr>
<tr>
<td></td>
<td>for a Web service input, or flowdata.Start/RequestRate/Country1 for a Web</td>
</tr>
<tr>
<td></td>
<td>service output field.</td>
</tr>
<tr>
<td>Web Service Input Field</td>
<td>This column displays all of the input fields for the port type and</td>
</tr>
<tr>
<td></td>
<td>operation specified when the WSDL file was selected. The fields in this</td>
</tr>
<tr>
<td></td>
<td>column are automatically populated. If you want to remove an input field,</td>
</tr>
<tr>
<td></td>
<td>click Mapping, expand the nodes of the sample document and de-select any</td>
</tr>
<tr>
<td></td>
<td>input fields that you want to remove.</td>
</tr>
<tr>
<td>Web Service Output Field</td>
<td>This column displays all of the output fields for the port type and</td>
</tr>
<tr>
<td></td>
<td>operation specified when the WSDL file was selected. The fields in this</td>
</tr>
<tr>
<td></td>
<td>column are automatically populated. If you want to remove an output field,</td>
</tr>
<tr>
<td></td>
<td>click Mapping, expand the nodes of the sample document and deselect any</td>
</tr>
<tr>
<td></td>
<td>output fields that you want to remove.</td>
</tr>
<tr>
<td>Mapping</td>
<td>Displays a hierarchical view of the sample document for the inputs to or</td>
</tr>
<tr>
<td></td>
<td>outputs from the Web service. You can use this feature to deselect input or</td>
</tr>
<tr>
<td></td>
<td>output fields (by default, all Web service input and output fields are</td>
</tr>
<tr>
<td></td>
<td>selected).</td>
</tr>
</tbody>
</table>

8.11.3 E-Mail Notification

Not supported with this activity.
8.12 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.

- Section 8.12.1, “Properties,” on page 213
- Section 8.12.2, “Data Item Mapping,” on page 213
- Section 8.12.3, “E-Mail Notification,” on page 214

8.12.1 Properties

The Entitlement activity has the following properties:

Table 8-19 Entitlement Activity Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Provides a name for the activity.</td>
</tr>
<tr>
<td>Set Workflow Status</td>
<td>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 Set Default Completion Status to Approved parameter (see Table 5-1, “Overview Properties,” on page 105) or the Approval Status activity (see Section 8.8, “Workflow Status,” on page 205).</td>
</tr>
</tbody>
</table>

8.12.2 Data Item Mapping

To bind the data items associated with the Entitlement activity, you define mappings for several DirXML® attributes.
Table 8-20  Entitlement Activity Data Item Mappings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Expression</td>
<td>Specifies a source expression for a DirXML mapping. When you click a cell in the Source Expression column, the ECMA expression builder displays to help you define your expression.</td>
</tr>
</tbody>
</table>

The DirXML mappings for the Entitlement are described below:

- 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:
  'CN=Groups,CN=GroupEntitlementLoopback,CN=TestDrivers,O=novell'
  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:
  '\MYTREE\novell\idmsample-doc\groups\Sales'
- 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 267.

8.12.3  E-Mail Notification

Not supported with this activity.

8.13  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 183).

A workflow must have at least one Entitlement or Entity activity.

- Section 8.13.1, “Properties,” on page 214
- Section 8.13.2, “Data Item Mapping,” on page 215
- Section 8.13.3, “E-Mail Notification,” on page 217

8.13.1  Properties

The Entity activity has the following properties:


<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Provides a name for the activity.</td>
</tr>
<tr>
<td>Entity Type</td>
<td>Specifies the target entity type: User or Group.</td>
</tr>
<tr>
<td>Operation</td>
<td>Indicates what kind of operation will be performed on the target entity:</td>
</tr>
<tr>
<td></td>
<td>• Create/Modify</td>
</tr>
<tr>
<td></td>
<td>• Delete attributes/values</td>
</tr>
<tr>
<td></td>
<td>• Delete entity</td>
</tr>
<tr>
<td></td>
<td>To create or modify attributes of an entity or to create a new entity,</td>
</tr>
<tr>
<td></td>
<td>select create/modify.</td>
</tr>
<tr>
<td></td>
<td>To delete attributes of an entity, select delete.</td>
</tr>
<tr>
<td></td>
<td>To delete an entity, select delete object.</td>
</tr>
<tr>
<td>Set Workflow Status</td>
<td>Specifies the approval status of the provisioning request. Set to True</td>
</tr>
<tr>
<td></td>
<td>for approved; otherwise, set to False. This method of setting workflow</td>
</tr>
<tr>
<td></td>
<td>status overrides other methods.</td>
</tr>
<tr>
<td></td>
<td>For example, the Set Default Completion Status to Approved parameter</td>
</tr>
<tr>
<td></td>
<td>(see Table 5-1, &quot;Overview Properties,&quot; on page 105) or the Approval</td>
</tr>
<tr>
<td></td>
<td>Status activity (see Section 8.8, &quot;Workflow Status,&quot; on page 205).</td>
</tr>
</tbody>
</table>

### 8.13.2 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 8-22  Entity Activity Data Item Mappings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity dn</td>
<td>Identifies the entity that is the target of the operation. The default value is recipient. To create a new object, specify a distinguished name that does not yet exist.</td>
</tr>
</tbody>
</table>

**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. After 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 `flowdata.get('groupdn')`; For details on using the DNMaker control, see Section 6.5.9, "DNMaker," on page 139. |
| Modify Type  | Indicates how the mapping should be performed for an attribute. The choices are |
|              | - Append Value                |
|              | - Replace Value               |
|              | - Replace All Values          |

For many attributes, *Replace Value* is the only option that makes sense; therefore, this option is selected automatically and cannot be changed.

You must specify the *Modify Type* setting before specifying the *Modify Value Expression* setting.
### Modify Value Expression

Specifies a source expression for an attribute. When you click a cell in the Modify Value Expression 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.

Designer automatically inserts a sample ECMAScript expression into this field. The code provided varies depending on the Operation property specified in Properties and the Modify Type selected in Data Item Mapping. For example, if you have specified Create/Modify for Operation and Replace All Values for Modify Type, Designer inserts an expression that helps you to create a vector:

```javascript
function list() { v=new java.util.Vector(); v.add('{Enter Item 1}'); v.add('{Enter Item 2}'); return v; }; list();
```

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 90) to store multiple attribute values, and use the getObject function to retrieve the values of the flowdata variable (see “ECMAScript Variables” on page 270).

**NOTE:** The cells in the Target Attribute column are not editable.

### 8.13.3 E-Mail Notification

Not supported with this activity.
This section provides details about working with Integration activities. Topics include:

- Section 9.2, “Adding an Integration Activity,” on page 219
- Section 9.3, “Moving Data to and from the Integration Activity,” on page 221
- Section 9.4, “Using the Integration Activity Editor Interface,” on page 224
- Section 9.5, “Actions,” on page 237

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 99).

2. Create a workflow for the provisioning request definition (see Chapter 7, “Creating the Workflow for a Provisioning Request Definition,” on page 165).

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 don’t need to concern yourself with the action model. You 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 8.11, “Integration Activity,” on page 210).

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 109). For example, if you are working with a Web service that provides stock quotes, you 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 90 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 221.

The integration activity automatically selects all of the input field associated with the port type and operation. You can remove the input fields or modify properties of the input fields by following this procedure:

4a Click Mapping.

The Sample Document dialog box is displayed.

4b Expand the nodes of the sample document and deselect any input fields that you want to remove.
4c If an input field is an unbounded element, right-mouse and select *Occurs*.

4d Type the maximum number in the *Enter Repeats* dialog, and click *OK*.

4e Click *OK* to return to the *Data Item Mapping* view.

5 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.
6 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.

![ECMA Expression Builder](image)

7 Click *OK* to return to the *Data Item Mapping* view.

8 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 221.

9 The Integration activity automatically selects all of the output fields associated with the port type and operation. If you want to remove some of the output fields, follow these steps:

9a Click *Mapping*.

The *Sample Document* dialog box is displayed.

9b Expand the nodes of the sample document and deselect any attributes that you want to remove.

9c Click *OK* to return to the *Data Item Mapping* view.

10 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.

11 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.

12 Click *OK* to close the ECMA expression builder.

Now you can 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.
Section 9.4.1, “XML Views,” on page 225
Section 9.4.2, “Action Model,” on page 229
Section 9.4.3, “WSDL Editor,” on page 236
Section 9.4.4, “Messages,” on page 236
Section 9.4.5, “Regenerating Code for the Action Model,” on page 236
Section 9.4.6, “Adding Actions to the Action Model,” on page 236

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” on page 225
- “Source View” on page 228

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.
Tree View Editing Features

The Tree view provides the following editing features:

- You can edit attribute values, attribute name, namespace names and values, text, and comments.
- You can insert new nodes by 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 259 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 you clicked.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove</td>
<td>Removes the selected item.</td>
</tr>
<tr>
<td>Add DTD Information</td>
<td>Displays a dialog box that you use to add DTD information. You can edit the Root Element Name, Public ID, and System ID.</td>
</tr>
</tbody>
</table>
Tree View Toolbar

Tree view toolbars provide the following features:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit Namespaces</td>
<td>Displays a dialog box that you use to add namespace declarations.</td>
</tr>
<tr>
<td>Add Attribute</td>
<td>Displays a dialog box that you use to define a new attribute.</td>
</tr>
<tr>
<td>Add Child</td>
<td>Displays a submenu with the following options:</td>
</tr>
<tr>
<td>Add Before</td>
<td>Comment</td>
</tr>
<tr>
<td>Add After</td>
<td>Add Processing Instruction</td>
</tr>
<tr>
<td></td>
<td>#PCDATA</td>
</tr>
<tr>
<td></td>
<td>CDATA Section</td>
</tr>
<tr>
<td></td>
<td>New Element</td>
</tr>
<tr>
<td>Replace with</td>
<td>Replaces the selected item with an item selected from the menu.</td>
</tr>
</tbody>
</table>

### Table 9-2  Tree View Toolbar

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Expand" /></td>
<td>Expands all nodes in the document.</td>
</tr>
<tr>
<td><img src="image" alt="Collapse" /></td>
<td>Collapses all nodes in the document.</td>
</tr>
<tr>
<td><img src="image" alt="Attach" /></td>
<td>Attaches a schema or DTD (see “Attaching a Schema or DTD” on page 227).</td>
</tr>
<tr>
<td><img src="image" alt="Help" /></td>
<td>Displays online help.</td>
</tr>
</tbody>
</table>

### 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 ![Attach](image) 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**.

![XML Schema dialog box](image1)

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**.

![DTD dialog box](image2)

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” on page 229
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

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td>Reverses the last action.</td>
</tr>
<tr>
<td>Redo</td>
<td>Reverses an undo operation.</td>
</tr>
<tr>
<td>Cut</td>
<td>Cuts the selected text to the clipboard.</td>
</tr>
<tr>
<td>Copy</td>
<td>Copies the selected text to the clipboard.</td>
</tr>
<tr>
<td>Paste</td>
<td>Pastes the clipboard contents at the insertion point.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the selected text.</td>
</tr>
<tr>
<td>Select All</td>
<td>Selects all of the text in the document.</td>
</tr>
<tr>
<td>Find</td>
<td>Displays a dialog box that you use to find and replace text within the document.</td>
</tr>
</tbody>
</table>

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 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 236), 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 236).

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 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 by using the Messages tab (see Section 9.4.4, “Messages,” on page 236), 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 is also 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 240). 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 is 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” on page 232
- “Finding and Replacing Text in the Action Model” on page 232

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 237), toggle breakpoints in the action model (see “Animation” on page 233) and perform other tasks.

Finding and Replacing Text in the Action Model

You can replace a word or string by using the *Replace* command on the action model menu.

1. Right-click in the action model, then select *Replace*.

2. Specify 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 do 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>
<thead>
<tr>
<th>Animation Toolbar Button</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td></td>
<td>Executes the action model.</td>
</tr>
<tr>
<td>Execute Current Action</td>
<td></td>
<td>Executes the currently selected action.</td>
</tr>
<tr>
<td>Start Animation</td>
<td></td>
<td>Starts the animation process. Enables <em>Step Into, Step Over, and Run to Breakpoint/End</em>.</td>
</tr>
<tr>
<td>End Animation</td>
<td></td>
<td>Stops the animation process.</td>
</tr>
</tbody>
</table>
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 can 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 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 selected 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 234).
2 Click *Step Into*. The first action in the action model is selected.
3 Click *Step Into* again. The selected action executes and the next action is selected.
4 Continue to work through the action model by clicking *Step Into* after each action executes and the subsequent action is selected.

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, because you can avoid tediously stepping through individual actions.

1 Start the animation (see “Starting Animation” on page 234).
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 is selected. 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. After 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 by using the Tree view and Source view editing features (see “Tree View” on page 225 and “Source View” on page 228).

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 225 and “Source View” on page 228). You can use this feature for entering test data that is 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:

![Regenerate Button](image)

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 by using any of the following methods. The new action is inserted below the line you selected.

- 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 233.

### 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 237
- Section 9.5.2, “Data Exchange,” on page 243
- Section 9.5.3, “Repeat,” on page 248
- Section 9.5.4, “Comment,” on page 255
- Section 9.5.5, “Decision,” on page 256
- Section 9.5.6, “Function,” on page 257
- Section 9.5.7, “Log,” on page 258
- Section 9.5.8, “Map,” on page 259

### 9.5.1 Advanced

This section includes descriptions of the following actions:

- “Apply Namespaces” on page 237
- “Throw Fault” on page 240
- “Try/On Fault” on page 242

**Apply Namespaces**

This section includes the following topics:

- “About the Apply Namespaces Action” on page 238
About the Apply Namespaces Action

Ideally, an Integration activity always receives 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 is 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 if 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 XPaths 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 259) 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 256) 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, resolves 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 240
- “Adding a Throw Fault Action” on page 241

**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 242. 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 by 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 is 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 267).
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 is 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 242
- “Adding a Try/On Fault Action” on page 242

#### 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.

Using 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 is 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-arrow 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.

9.5.2 Data Exchange

This section includes descriptions of the following actions:

- “WS Interchange” on page 244
- “XML Interchange” on page 245
WS Interchange

This section includes the following topics:

- “About the WS Interchange Action” on page 244
- “Adding a WS Interchange Action” on page 244

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 automatically creates a WS Interchange action when it creates the action model.

In most cases there is 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 is inserted below the line that you selected).
2. Select New Action > Data Exchange > WS Interchange.

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.

![Image of WS Interchange window]

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 is 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 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 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 246
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*. 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 is 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

Because 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.
If you want 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.

Select the nodes that you want to keep in the document.

Nodes that are not selected are discarded before parsing the DOM.

When you have selected nodes that you want to keep, click *OK* to return to the *XML Interchange* dialog box.

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 249
- “Continue” on page 249
- “Declare Group” on page 250
- “Repeat for Element” on page 251
- “Repeat for Group” on page 253
- “Repeat While” on page 254
Break

This section includes the following topics:

- “About the Break Action” on page 249
- “Adding a Break Action” on page 249

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 is 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 249
- “Adding a Continue Action” on page 249

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 250
- “Adding a Declare Group Action” on page 250

**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 is 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 want 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 251
- “Adding a Repeat for Element Action” on page 252

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 is inserted below the line that you selected).

2 Select New Action > Repeat > Repeat for Element.

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 “S1Lineitem.”

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 Select Always create new output elements if you have repeating actions that should add new elements rather than updating existing elements.
Specify an XPath expression, or click the ECMA expression builder button and build an XPath expression for the repeating element.

Click OK. The Repeat for Element loop is added to the action model.

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 253
- “Adding a Repeat for Group Action” on page 253

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 a 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 the source group used in the Repeat for Group dialog boxes are the list names created by the Declare Group action. The list names perform two functions. They are an alternate name or shorthand 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 shorthand 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 is 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 254
- “Adding a Repeat While Action” on page 255

**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 shorthand for the XPath target of any Map actions within the loop. This saves you the work of respecifying 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, Repeat While does not need 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 is inserted below the line that you selected).

2. Select *New Action > Repeat > Repeat While*.

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 to 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 255
- “Adding a Comment Action” on page 255

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 is 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 256
- “Adding a Decision Action” on page 256

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 is 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.

```xml
<when xPath="getQuoteSoapIn/getQuote/symbol">testString!="ECME"
  <true>
     ...</true>
  <false>
     ...</false>
</when>
```

5 In the action model pane, select the \textit{TRUE} branch.

6 Add one or more actions that will execute if the expression is True.

7 Select the \textit{FALSE} branch.

8 Add one or more actions to execute if the expression is False.

You can nest other Decision actions inside the \textit{TRUE} or \textit{FALSE} branches of the Decision action.

### 9.5.6 Function

- “About the Function Action” on page 257
- “Adding a Function Action” on page 257

#### 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 is inserted below the line that you selected).

2 Select \textit{New Action > Function}.

3 Type the function in the \textit{Function Expression} field or click the ECMA expression builder button to build an ECMAScript expression.

4 Click \textit{OK}.
9.5.7 Log

- “About the Log Action” on page 258
- “Adding a Log Action” on page 258

**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 is inserted below the line that you selected).

![Log settings](image)

3. Select the type of log that you want 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.
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 is 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 is 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 259
- “Adding a Map Action” on page 261
- “Advanced Mapping Options” on page 261
- “Transforming Elements with the Content Editor” on page 264

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 shorthand for an XPath expression.

- “Default Mapping Behavior” on page 260
- “Leaf Elements That Contain Markup” on page 260

### 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>
<thead>
<tr>
<th>Map Type</th>
<th>Default Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leaf Element to Leaf Element</td>
<td>Transfers only the element data.</td>
</tr>
<tr>
<td>Leaf Element to Branch Element</td>
<td>Transfers only the element data.</td>
</tr>
<tr>
<td>Branch Element to Leaf Element</td>
<td>Transfers the entire branch, including all child elements and attribute data under the branch.</td>
</tr>
<tr>
<td>Branch Element to Branch Element</td>
<td>Transfers the entire branch, including all child elements and attribute data under the branch after removing the target's current branch.</td>
</tr>
<tr>
<td>A particular Leaf Element in a list of Leaf Elements, to Element</td>
<td>Transfers the element data from the selected leaf (or element instance) to the target element.</td>
</tr>
<tr>
<td>Attribute to Attribute</td>
<td>Transfers only the attribute data.</td>
</tr>
<tr>
<td>Element to Attribute</td>
<td>Transfers element data to attribute data.</td>
</tr>
<tr>
<td>Attribute to Element</td>
<td>Transfers only the attribute data.</td>
</tr>
</tbody>
</table>

Many of these behaviors can be altered, on an action-by-action basis, through the use of the Advanced mapping dialog box (see “Advanced Mapping Options” on page 261).

### 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 the Source view, you’ll see that markup characters typed into a node are converted to entities. For example, a “<” character is converted to `<alt;`.
**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 is 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 261) or *Content Editor* (see “Transforming Elements with the Content Editor” on page 264) 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.
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 262
- “Deep Copy” on page 263
- “Create Target” on page 263
- “Create Target as CDATA Section” on page 263

Copy Attributes

You use Copy Attributes to specify how attributes are mapped. Copy Attributes has the following options:

Table 9-6  Copy Attributes Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>For Non-leaf Root Nodes and Dependents</td>
<td>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.</td>
</tr>
<tr>
<td>Never</td>
<td>Specifies that no attribute data (whether for parent or leaf nodes) is carried over during mapping.</td>
</tr>
</tbody>
</table>
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 261), 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 might have specified $Output/Root/MyParentNode/SomeOtherElement

If the incoming Input document does not have a node corresponding to Root/MySourceElement, the Integration activity by default creates 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>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only if Source exists</td>
<td>Specifies that the Map Action is 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.</td>
</tr>
<tr>
<td>Raise Error</td>
<td>Specifies that if the input document doesn’t contain the node specified in the Source XPath, it is 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.</td>
</tr>
<tr>
<td>Always</td>
<td>Specifies that the target node should always be created (the default behavior). When Always is selected, you can use the Default Value field to specify a default data value for the target element.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only if source contains markup</td>
<td>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 is placed, unmodified, in a CDATA section in the target DOM. This is the default behavior.</td>
</tr>
<tr>
<td>Never</td>
<td>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 &gt; for &gt;, on the output side.</td>
</tr>
<tr>
<td>Always</td>
<td>Specifies that whatever form the source data takes, it will get wrapped in a CDATA section on output.</td>
</tr>
</tbody>
</table>

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 261), 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 want 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.
Working with ECMA Expressions

This section provides details on using the ECMA expression builder. Topics include:

- Section 10.1, “About the ECMA Expression Builder,” on page 267
- Section 10.2, “ECMAScript Examples,” on page 277
- Section 10.3, “ECMAScript API,” on page 280

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 workflow path by using the Condition activity.
  - Write different log messages that are conditionally triggered by using a single Log activity.

This section describes some of the techniques and capabilities applicable to the use of scripting.

- Section 10.1.1, “About ECMAScript,” on page 268
- Section 10.1.2, “ECMAScript Capabilities,” on page 268
- Section 10.1.3, “Using the ECMA Expression Builder,” on page 268
- Section 10.1.4, “About Java Integration,” on page 274
- Section 10.1.5, “About XPath Integration,” on page 275
- Section 10.1.6, “Performance Considerations,” on page 276

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 189. For descriptions of the system variables available in a workflow, see Section 4.3.3, “Understanding Workflow Data,” on page 90.
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.


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), 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.
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 269
- “Selecting a DN” on page 270
- “ECMAScript Variables” on page 270
- “Functions/Methods” on page 270
- “ECMAScript Operators” on page 271
- “VDX Expr” on page 273
- “Using Special Characters” on page 273

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 dimmed (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 90.

The ECMA expression builder provides two methods for reading flowdata variables.

Table 10-1  Methods for Reading Flowdata Variables

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flowdata.get(variable-name)</td>
<td>Returns a string as the node value for a variable (representing an XPath expression) in the workflow document.</td>
</tr>
<tr>
<td>flowdata.getObject(variable-name)</td>
<td>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.</td>
</tr>
</tbody>
</table>

Functions/Methods

For a description of the functions and methods available in the ECMA expression builder, see Section 10.3, “ECMAScript API,” on page 280.
ECMAScript Operators

The following tables provide descriptions of the operators supported by the ECMA expression builder.

**Table 10-2 Math**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Add</td>
<td>Returns the sum of two numerical values (either literals or variables).</td>
</tr>
<tr>
<td>- Subtract</td>
<td>Subtracts one number from another.</td>
</tr>
<tr>
<td>* Multiply</td>
<td>Returns the product of two numerical values (either literals or variables).</td>
</tr>
<tr>
<td>/ Divide</td>
<td>Divides one number by another.</td>
</tr>
</tbody>
</table>

**Table 10-3 Assignment**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>= Assignment</td>
<td>Assigns the value of the right operand to the left operand.</td>
</tr>
<tr>
<td>+= Add to this</td>
<td>Adds the left and right operands and assigns the result to the left operand. For example, a += b is the same as a = a + b.</td>
</tr>
<tr>
<td>-= Subtract from this</td>
<td>Subtracts the right operand from the left operand and assigns the result to the left operand. For example, a -= b is the same as a = a - b.</td>
</tr>
<tr>
<td>*= Multiply to this</td>
<td>Multiplies the two operands and assigns the result to the left operand. For example, a *= b is the same as a = a * b.</td>
</tr>
<tr>
<td>/= Divide this to</td>
<td>Divides the left operand by the right operand and assigns the result to the left operand. For example, a /= b is the same as a = a / b.</td>
</tr>
<tr>
<td>%= Modulus</td>
<td>Divides the left operand by the right operand and assigns the remainder to the left operand. For example, a %= b is the same as a = a % b.</td>
</tr>
<tr>
<td>&amp;= Apply bitwise AND to this</td>
<td>Performs bitwise AND on operands and assigns the result to the left operand. For example, a &amp;= b is the same as a = a &amp; b.</td>
</tr>
<tr>
<td></td>
<td>= Apply bitwise OR to this</td>
</tr>
<tr>
<td>&lt;&lt;= Apply bitwise left shift to this</td>
<td>Performs bitwise left shift on operands and assigns the result to the left operand. For example, a &lt;&lt;= b is the same as a = a &lt;&lt; b.</td>
</tr>
<tr>
<td>&gt;&gt;= Apply bitwise signed right shift to this</td>
<td>Performs bitwise right shift on operands and assigns the result to the left operand. For example, a &gt;&gt;= b is the same as a = a &gt;&gt; b.</td>
</tr>
</tbody>
</table>
### Table 10-4  Other

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;&gt;= Apply bitwise unsigned right shift to this</td>
<td>Performs bitwise unsigned right shift on operands and assigns the result to the left operand. For example, ( a &gt;&gt;&gt;= b ) is the same as ( a = a &gt;&gt; b ).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Modulus</td>
<td>Divides the left operand by the right operand and returns the integer remainder.</td>
</tr>
<tr>
<td>++ Autoincrement</td>
<td>Increments the operand by one (can be used before or after the operand).</td>
</tr>
<tr>
<td>-- Autodecrement</td>
<td>Decrements the operand by one (can be used before or after the operand).</td>
</tr>
<tr>
<td>~ Bitwise NOT</td>
<td>Inverts the bits of its operand.</td>
</tr>
<tr>
<td>&amp; Bitwise AND</td>
<td>Returns a 1 in each bit position for which the corresponding bits of both operands are ones.</td>
</tr>
<tr>
<td></td>
<td>Bitwise OR</td>
</tr>
<tr>
<td>^ Bitwise XOR</td>
<td>Returns a 1 in each bit position for which the corresponding bits of either but not both operands are ones.</td>
</tr>
<tr>
<td>&lt;&lt;= Bitwise left shift</td>
<td>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.</td>
</tr>
<tr>
<td>&gt;&gt;= Signed bitwise right shift</td>
<td>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.</td>
</tr>
<tr>
<td>&gt;&gt;&gt;&gt; Unsigned bitwise right shift</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Table 10-5  Relational

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>== Equal</td>
<td>Assigns the value of the right operand to the left operand.</td>
</tr>
<tr>
<td>!= Not Equal</td>
<td>Returns a Boolean True if the operands are not equal.</td>
</tr>
<tr>
<td>&lt; Less than</td>
<td>Returns True if the left operand is less than the right operand.</td>
</tr>
<tr>
<td>&gt; Greater than</td>
<td>Returns True if the left operand is greater than the right operand.</td>
</tr>
</tbody>
</table>
This pane allows you to insert Entity definitions (see Section 3.2, “Working with Entities and Attributes,” on page 45) 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:
You also can specify any Unicode character by using `\u` followed immediately by four hexadecimal digits. Here are some examples:

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\b</code></td>
<td>Backspace</td>
</tr>
<tr>
<td><code>\f</code></td>
<td>Form feed</td>
</tr>
<tr>
<td><code>\n</code></td>
<td>New line</td>
</tr>
<tr>
<td><code>\r</code></td>
<td>Carriage return</td>
</tr>
<tr>
<td><code>\t</code></td>
<td>Horizontal tab</td>
</tr>
<tr>
<td><code>\&quot;</code></td>
<td>Double quote</td>
</tr>
<tr>
<td><code>\\</code></td>
<td>Backslash ()</td>
</tr>
<tr>
<td><code>\'</code></td>
<td>Apostrophe</td>
</tr>
</tbody>
</table>

### Table 10-9  Escape Sequence Examples for Unicode Characters

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\u00A3</code></td>
<td>Pound symbol (£)</td>
</tr>
<tr>
<td><code>\u20AC</code></td>
<td>Euro symbol (€)</td>
</tr>
</tbody>
</table>

#### 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 280.

### 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, use ECMAScript syntax to call your Java methods. The following example creates a vector:

```javascript
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:

```javascript
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).

**10.1.5 About XPath Integration**

This section includes the following topics:

- “XPath in Workflows” on page 275
- “XPath in Integration Activities” on page 275

**XPath in Workflows**

A provisioning request definition workflow supports a special object called `flowdata` (see Section 4.3.3, “Understanding Workflow Data,” on page 90). 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`:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>flowdata.{arguments}</code></td>
<td><code>flowdata.parent/child[1]</code></td>
</tr>
<tr>
<td><code>ECMAScript</code></td>
<td><code>flowdata.reason</code></td>
</tr>
<tr>
<td><code>XPath</code></td>
<td></td>
</tr>
</tbody>
</table>

To get an object from `flowdata`:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>flowdata.get{arguments}</code></td>
<td><code>flowdata.getObject('parent/child[1]')</code></td>
</tr>
<tr>
<td><code>ECMAScript</code></td>
<td><code>flowdata.get('reason')</code></td>
</tr>
<tr>
<td><code>XPath</code></td>
<td></td>
</tr>
</tbody>
</table>

For information about the `flowdata.get()` and `flowdata.getObject()` methods, see Table 10-1 on page 270.

**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:
When you use 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>
<thead>
<tr>
<th>XPath Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>The single forward slash represents an absolute path to an element. For example, /ABC selects the root element ABC.</td>
</tr>
<tr>
<td>//</td>
<td>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.</td>
</tr>
<tr>
<td>*</td>
<td>The asterisk selects all elements located by the preceding path. For example, <em>ABC/DEF selects all elements enclosed by elements ABC/DEF. //</em> selects all elements.</td>
</tr>
<tr>
<td>[]</td>
<td>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[&quot;Table&quot;] selects all elements that have the content “Table.”</td>
</tr>
<tr>
<td>@</td>
<td>The At sign selects elements with a specified attribute. For example, /ABC@name selects all elements in ABC that have an attribute called name.</td>
</tr>
<tr>
<td></td>
<td>The vertical bar allows you to specify multiple paths. For example, //ACB</td>
</tr>
<tr>
<td>$</td>
<td>The dollar sign allows you to reference other documents besides the current one: INVOICEBATCH/INVOICE[Seller/NAME=$PROJECT/USERCONF/GNAME]</td>
</tr>
<tr>
<td>function()</td>
<td>XPath has numerous functions that you can add to your XPath addresses. For example, //<em>[count(</em>)=2] selects all elements that have two children.</td>
</tr>
<tr>
<td>math operator()</td>
<td>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.</td>
</tr>
</tbody>
</table>

You can find the complete list of operators in the W3 Recommendation XML Path Language (XPath) (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, such as 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 creating 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:

```javascript
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.

- Section 10.2.1, “General Examples,” on page 277
- Section 10.2.2, “Flowdata Examples,” on page 278
- Section 10.2.3, “Form Control Examples,” on page 278
- Section 10.2.4, “Error Handling,” on page 279

### 10.2.1 General Examples

To create a function in the ECMA expression builder, create the function inline:

```javascript
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” on page 278
- “Creating an XML Element with Child Element and Adding it to the Flowdata” on page 278

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](image)

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 you can 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” on page 278
- “Getting an Individual Value from a Multivalued Control” on page 279
- “Populating a List or Checkbox Item” on page 279
- “Comparing DNs” on page 279

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:

```javascript
flowdata.colors
```

To get a value from `colors` (for example, the first value), use the following expression on a downstream activity:

```javascript
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:

```javascript
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:

```javascript
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

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:

```javascript
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.'; }
```
10.3 ECMAScript API

This section includes the following topics:

- Section 10.3.1, “Form Action Script Methods,” on page 280
- Section 10.3.2, “DOM Methods,” on page 290
- Section 10.3.3, “ECMAScript Core,” on page 314
- Section 10.3.4, “Global Functions,” on page 333
- Section 10.3.5, “IDVault Functions,” on page 333

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 280
- “Field” on page 285
- “Event” on page 288
- “Lists” on page 289
- “Queries” on page 289
- “Container” on page 289

Form

Lets you work with Form methods. This section includes the following methods:

- “focus(fieldname)” on page 281
- “select(fieldname)” on page 281
- “activate(fieldname)” on page 281
- “setRequired(fieldname, is-required)” on page 281
- “InterceptAction(actionname, order, function)” on page 281
- “getLocale()” on page 282
- “getRBMessage()” on page 282
- “stringToDate()” on page 282
- “dateToString()” on page 282
- “isValidDate(date)” on page 282
- “isValidDate(date,include-time)” on page 283
focus(fieldname)
    form.focus(fieldname)

Sets the focus to the specified field. For list-based or choice-based controls, sets the focus to either the selected choice or when no selection is made, it sets the focus to the first choice. If a fieldname parameter is passed and that field is list or choice based, it sets the focus on the choices corresponding to the values parameter. If the value is an array, only the first value is used to determine on which check box or radio button to set focus. If the specified field is invisible or disabled, this method has no effect.

select(fieldname)
    form.select(fieldname)

If no values parameter is passed in, this method sets the focus to the underlying text field. For list-based or choice-based controls, this method sets the focus to the selected choice or if no selection was made, to the first choice. If a values parameter is passed and the field is list or choice based it will set the focus on the choices corresponding to the fieldname parameter. This method has no effect on disabled or invisible fields.

activate(fieldname)
    form.activate(fieldname)

A combination of setFocus() and select().

setRequired(fieldname, is-required)
    form.setRequired("fieldname", is-required)

Sets the field to required if is-required is True or optional otherwise. A field that is required blocks the form submission if it is empty.

InterceptAction(actionname, order, function)
    form.interceptAction("actionname", "order", "function")

Allows you to intercept the script attached to an action button. The function passed in is executed based on the order parameter.

Valid actionname values are SubmitAction and CancelAction.
The choices for actionname for an approval form are: ApprovalAction, RefusalAction, DenyAction, UpdateAction, CancelAction and CommentAction.

Valid values for the advice parameter are:

Before: The function is called before the script attached to the button executes.

after: The function is called after the script attached to the button executes.

around: The function is passed a parameter that allows you to decide whether to execute the script attached to the button. The following example shows the submit action intercepted. The form is only submitted if the user replies Yes.

```javascript
window.inv = function (invocation) {
    if (confirm("Are you sure you want to submit ?")) {
        var result = invocation.proceed();
        return result;
    }
};
form.interceptAction("SubmitAction", "around", window.inv);

getLocale()
form.getLocale()

Returns the current locale. Can be used as input for all methods that support a locale parameter.

getRBMessage()
form.getRBMessage(key)
form.getRBMessage(key, value[s])
form.getRBMessage(key, value[s], bundle)

This method tries to find an entry with key in the resource bundle with ID bId. The resourcebundle Java class should extend the java.util.ListResourceBundle.Parameter. The param can be used to pass in replacements for parameters (\{0\}, \{1\}, etc) in message "msg"; for example:

```javascript
var msg = frm.getRBMessage("mykey", ["value0", "value1"], "mybundle");
```

stringToDate()
form.stringToDate(date)
form.stringToDate(date, include-time)

Converts a date string to a Date. The format must correspond to the dateform for the current locale, as used in the DatePicker. The value of a DatePicker control can be converted with this method.

Example:

```javascript
form.showMsg("Date=\"+
form.stringToDate(d, true));
```

dateToString()
form.dateToString(date)
form.dateToString(date, include-time)

Converts a date to a string that can be stored in the DatePicker, for example:

```javascript
var d = form.dateToString(new Date(), true);
form.setValues("hireDate", d);
```

isValidDate(date)
form.isValidDate(date)
Use this to validate the correct format for a date string.

**isValidDate(date, include-time)**

```
form.isValidDate(date, include-time)
```

Use this to validate the correct format for a date string.

**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 (because of 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 are 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"]);
```

```javascript
showFatal(string)
```

```
form.showFatal("my fatal" {0},{1}"", ["value0","value1");
```

Adds a 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 (because of 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 are 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");
```

```javascript
enable(fieldname)
```

```
form.enable("fieldname")
```

Enables a field on a form.

```javascript
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:

```javascript
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:

- “activate()” on page 286
- “disable()” on page 286
- “enable()” on page 286
- “fireEvent()” on page 286
- “focus()” on page 286
- “getLabel()” on page 286
- “getName()” on page 286
- “getValue()” on page 287
- “hide()” on page 287
- “getValues()” on page 287
activate()
field.activate(value[s])

This method is a combination of field.focus() and field.select().

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.

enable()
field.enable()

Enable 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()).

focus()
field.focus(value[s])

If no values parameter is passed in, this method sets the focus to the underlying text field. For list-based or choice-based controls, this method sets the focus to the selected choice if no selection was done to the first choice. If a values parameter is passed and if the field is list-based or choice-based, this method sets the focus on the choices corresponding to the values parameter. If the values is an array, only the first value is used to determine the check box or radio button to set focus. This method has no effect on disabled or invisible fields.

getLabel()
field.getLabel()

Gets the label associated with the field. If no label is found, this method returns the name of the field.

getName()
field.getName()

Gets the name of the field.
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.

hide()
field.hide()

Hides this field.

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.

show()
field.show()

Shows this field.

select()
field.select(value[s])

If no values parameter is passed in, this method sets the focus to the underlying text field. For list-based or choice-based controls, it sets the focus to either the selected choice or if no selection was made, it sets the focus to the first choice. If a values parameter is passed and if the field is list-based or choice-based it sets the focus on the choices corresponding to the values parameter. If the value parameter is an array, only the first value is used to determine which check box or radio button to set focus on. This method has no effect on disabled or invisible fields.

setRequired()
field.setRequired(is-required)

Sets the field to required if isRequired is True or optional otherwise. A field that is required blocks the form submission if it is empty.

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 an onload event.

NOTE: This method triggers the onchange event for the field.
Examples:

```javascript
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
```

`validate()`

```
field.validate()
```

Triggers browser validation for the field. To validate the data entered in this field as soon as the user navigates to another field, call this method in the onchange event. Returns True if validation errors were detected, False otherwise.

**Event**

Lets you work with events. This section includes the following methods:

- “getEventName()” on page 288
- “getOrigin()” on page 288
- “getValue()” on page 288
- “getValues()” on page 288

`getEventName()`

```
event.getEventName()
```

Returns the name of the event.

`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, because 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, because 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.getValue(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)**

IDVault.globalQuery("fieldname", "key", "param")

Executes the predefined directory abstraction layer query key (see “Queries General Properties” on page 77). 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 two entries, the first an array with the resulting DNs; the second entry an array with the display labels.
### Table 10-11  Container Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fieldname</code></td>
<td>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 <code>fieldname</code> parameter.</td>
</tr>
<tr>
<td><code>rootdn</code></td>
<td>If the <code>rootdn</code> parameter is empty, the root container for the default entity is used.</td>
</tr>
<tr>
<td><code>scope</code></td>
<td>If the <code>scope</code> parameter is empty, one-level is used. Valid choices for <code>scope</code> are “o” (onelevel) and “s” (subtree).</td>
</tr>
<tr>
<td><code>showdn</code></td>
<td>If the parameter <code>showDN</code> is true, the full DN is used for the display label. Otherwise the naming part (for example, ou, dc) is displayed.</td>
</tr>
</tbody>
</table>

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 290
- “Document” on page 295
- “Element” on page 299
- “Attribute” on page 305
- “CharacterData” on page 306
- “NodeList” on page 307
- “NamedNodeMap” on page 309
- “Text” on page 311
- “DocumentType” on page 311
- “DOMImplementation” on page 312
- “Notation” on page 313
- “Entity” on page 313
- “ProcessingInstruction” on page 314

#### Node

Lets you work with nodes. This section includes the following topics:

- “attributes” on page 291
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 with one of the following values:
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 XPath Type 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 subtree
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 the IsSupported Method**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature</td>
<td>Core</td>
</tr>
<tr>
<td></td>
<td>XML</td>
</tr>
<tr>
<td></td>
<td>HTML</td>
</tr>
<tr>
<td></td>
<td>Views</td>
</tr>
<tr>
<td></td>
<td>Stylesheets</td>
</tr>
<tr>
<td></td>
<td>CSS</td>
</tr>
<tr>
<td></td>
<td>CSS2</td>
</tr>
<tr>
<td></td>
<td>Events</td>
</tr>
<tr>
<td></td>
<td>UIEvents</td>
</tr>
<tr>
<td></td>
<td>MouseEvents</td>
</tr>
<tr>
<td></td>
<td>MutationEvents</td>
</tr>
<tr>
<td></td>
<td>HTMLEvents</td>
</tr>
<tr>
<td></td>
<td>Range</td>
</tr>
<tr>
<td></td>
<td>Transversal</td>
</tr>
</tbody>
</table>
Example:
```
aNodeObject.isSupported("Core","2.0")
```

Document

Lets you work with documents. This section includes the following topics:

- “doctype” on page 295
- “documentElement” on page 296
- “implementation” on page 296
- “text” on page 296
- “createAttribute(name)” on page 296
- “createCDATASection(data)” on page 296
- “createComment(data)” on page 296
- “createDocumentFragment()” on page 296
- “createElement(tagName)” on page 296
- “createEntityReference(name)” on page 296
- “createProcessingInstruction(target,data)” on page 296
- “createTextNode(data)” on page 297
- “getElementsByTagName(tagName)” on page 297
- “reset()” on page 297
- “setDTD(Node RootElementName, Object PublicName, Object URL)” on page 297
- “setValue(Object aValue)” on page 297
- “toString()” on page 297
- “XPath(String asPattern)” on page 297
- “importNode(sourceNode, deep)” on page 298
- “createElementNS(namespaceURI, qualifiedName)” on page 298
- “createAttributeNS(namespaceURI, qualifiedName)” on page 298
- “getElementsByTagNameNS(namespaceURI, localName)” on page 299
- “getElementById(elementId)” on page 299
- “setSkipNameSpaces(abFlag)” on page 299
- “setEncoding(encoding)” on page 299

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 the 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 the ImportNode Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceNode</td>
<td>The node to import.</td>
</tr>
<tr>
<td>deep</td>
<td>A Boolean. If True, recursively import the subtree under the specified node.</td>
</tr>
</tbody>
</table>

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 the createElementNS Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>A string representing the namespace URI that you want to create for the element.</td>
</tr>
<tr>
<td>qualifiedName</td>
<td>A string representing the name to create for the element. qualifiedName = namespaceprefix + : + localName</td>
</tr>
</tbody>
</table>

Example:
Temp.createElementNS("someURI", "nsprefix: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 the createAttributeNS Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>A string representing the namespace URI that you want to create for the attribute.</td>
</tr>
<tr>
<td>qualifiedName</td>
<td>A string representing the name to create for the attribute. qualifiedName = namespaceprefix + : + localName</td>
</tr>
</tbody>
</table>
Example:
Temp.createAttributeNS("someURI","nsprefix: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 the getElementsByTagNameNS Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>A string of the elements on which to match. The special value &quot;+*&quot; matches all namespaces.</td>
</tr>
<tr>
<td>qualifiedName</td>
<td>A string of the elements on which to match. The special value &quot;+*&quot; matches all local names.</td>
</tr>
</tbody>
</table>

Example:
Temp.getElementsByTagNameNS("someURI", "someName")

getElementById(elementId)
Element getElementById(elementId)

W3C DOM Level 2 Document method. Returns the Element for which the ID 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 300
- “text” on page 300
- “booleanValue()” on page 300
- “countOfElement(String propertyName)” on page 301
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()

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.

setTitle(String asText)

ECMAScript extension method. Sets the text node associated with this element.

toString()

ECMAScript extension method. Gets the text node associated with this element.

toNumber()

Number toNumber()

ECMAScript extension method. Gets the text node and converts it to a number.

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 the getAttributeNS Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>Specifies a string representing the namespace URI of the target Attr.</td>
</tr>
<tr>
<td>localName</td>
<td>Specifies a string of the localName of the target Attr.</td>
</tr>
</tbody>
</table>

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 the setAttributeNS Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>The namespace URI of the attribute to create or alter.</td>
</tr>
<tr>
<td>qualifiedName</td>
<td>Specifies the qualified name of the attribute to create or alter.</td>
</tr>
<tr>
<td>TIP: qualifiedName = namespaceprefix + :: + localName</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Specifies the value to set in string form.</td>
</tr>
</tbody>
</table>

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 the removeAttributeNS Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>Specifies the namespaceURI of the attribute to remove.</td>
</tr>
<tr>
<td>localName</td>
<td>Specifies the name of the attribute to remove.</td>
</tr>
</tbody>
</table>

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 the getAttributeNodeNS Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>Specifies the namespaceURI of the attribute to retrieve.</td>
</tr>
<tr>
<td>localName</td>
<td>Specifies the name of the attribute to retrieve.</td>
</tr>
</tbody>
</table>

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 the getElementsByTagNameNS Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>Specifies the namespaceURI of the elements on which to match. The special value &quot;*&quot; matches all namespaces.</td>
</tr>
<tr>
<td>localName</td>
<td>Specifies the localName of the elements on which to match. The special value &quot;*&quot; matches all local names.</td>
</tr>
</tbody>
</table>

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 the hasAttributeNS Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>Specifies the namespaceURI of the attribute for which to look.</td>
</tr>
<tr>
<td>localName</td>
<td>Specifies the localName of the attribute for which to look.</td>
</tr>
</tbody>
</table>

Example:
Temp.XPath("A/B[0]").hasAttributeNS("someURI", "someName")

Attribute

Lets you work with attributes. This section includes the following topics:

- "name" on page 306
- "specified" on page 306
- "text" on page 306
- "value" on page 306
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)
Designer extension method. Sets the value of an attribute from the passed object.

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 307
- “length” on page 307
- “appendData(arg)” on page 307
- “insertData(offset, arg)” on page 307
- “deleteData(offset, count)” on page 307
- “replaceData(offset, count, arg)” on page 307
- “substringData(offset, count)” on page 307
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 308
- “avg([NodeList])” on page 308
- “count([NodeList])” on page 308
- “item(index)” on page 308
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 is 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 for 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 are not 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 for 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 is 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 for 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 for 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 is 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 for 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 return 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 310
- “getNamedItem(name)” on page 310
- “getNamedItemNS(namespaceURI, localName)” on page 310
- “item(index)” on page 310
- “removeNamedItem(name)” on page 310
- “removeNamedItemNS(namespaceURI, localName)” on page 310
- “setNamedItem(arg)” on page 311
- “setNamedItemNS(Node arg)” on page 311
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 the NamedNodeMap Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>Specifies the namespaceURI of the node to retrieve.</td>
</tr>
<tr>
<td>localName</td>
<td>Specifies the localName of the node to retrieve.</td>
</tr>
</tbody>
</table>

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 the removeNamedItemNS Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>Specifies the namespaceURI of the node to remove.</td>
</tr>
</tbody>
</table>
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 311
- “entities” on page 312
- “internalSubset” on page 312
- “notations” on page 312
- “publicId” on page 312
- “systemId” on page 312

**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 312
- “createDocumentType(qualifiedName, publicID, systemID)” on page 312
- “hasFeature(feature, version)” on page 313

createDocument(namespaceURI, qualifiedName, doctype)

Document createDocument(namespaceURI, qualifiedName, doctype)


Table 10-25 Parameters for the DOMImplementation Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaceURI</td>
<td>Specifies the namespaceURI of the document element to create.</td>
</tr>
<tr>
<td>qualifiedName</td>
<td>Specifies the qualified name of the document element to create.</td>
</tr>
<tr>
<td>doctype1</td>
<td>Specifies the type of document to create, or null.</td>
</tr>
</tbody>
</table>

createDocumentType(qualifiedName, publicID, systemID)

DocumentType createDocumentType(qualifiedName, publicID, systemID)
Table 10-26  Parameters for the createDocumentType Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qualifiedName</td>
<td>Specifies the qualified name of the document element to create. qualifiedName = namespaceprefix + : + localName</td>
</tr>
<tr>
<td>publicID</td>
<td>Specifies the external subset public identifier.</td>
</tr>
<tr>
<td>systemID</td>
<td>Specifies the external subset system identifier.</td>
</tr>
</tbody>
</table>

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 313
- “systemId” on page 313

**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 313
- “systemId” on page 314
- “notationName” on page 314

**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 314
- “data” on page 314

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 314
- “Boolean Object” on page 315
- “Date Object” on page 316
- “Function Object” on page 322
- “Global” on page 322
- “Math Object” on page 324
- “Number Object” on page 328
- “Object” on page 330
- “String Object” on page 330

Array Object

Lets you work with arrays. This section includes the following topics:

- “Array(item0, item1, . . .)” on page 315
- “join(separator)” on page 315
- “length” on page 315
- “reverse()” on page 315
“sort(comparefn)” on page 315
“toString()” on page 315

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 315
“toString()” on page 316
“valueOf()” on page 316

Boolean()
Boolean( [true/false] )

Constructor. Optionally takes either 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 317
- “getDate()” on page 317
- “getDay()” on page 317
- “getFullYear()” on page 317
- “getHours()” on page 317
- “getMilliseconds()” on page 317
- “getMinutes()” on page 318
- “getMonth()” on page 318
- “getSeconds()” on page 318
- “getTime()” on page 318
- “getTimezoneOffset()” on page 318
- “getUTCDate()” on page 318
- “getUTCDay()” on page 318
- “getUTCFullYear()” on page 318
- “getUTCHours()” on page 318
- “getUTCMilliseconds()” on page 318
- “getUTCMilliseconds()” on page 318
- “getUTCMilliseconds()” on page 318
- “getUTCMilliseconds()” on page 319
- “getUTCMilliseconds()” on page 319
- “getUTCMilliseconds()” on page 319
- “getUTCMilliseconds()” on page 319
- “getFullYear()” on page 319
- “parse(string)” on page 319
- “setDate(date)” on page 319
- “setFullYear(year[,mon[,date]])” on page 319
- “setHours(hour[,min[,sec[,ms]]])” on page 319
- “setMilliseconds(ms)” on page 319
- “setMinutes(min[,sec[,ms]])” on page 319
- “setMonth(mon[,date])” on page 320
- “setSeconds(sec [, ms ])” on page 320
- “setTime(time)” on page 320
The constructor of the Date can have various signatures. The date constructor format can accept up to seven parameters, in the following format: new Date(year,month,date,hrs,mins,secs,ms). This date must be a java.util.Date object and not an ECMAScript Date object if you intend to use it with the Identity Manager User Application workflow system.

**getDate()**

getDay()

Returns DateFromTime(LocalTime(t)).

**getDay()**

day()

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()

gMonth()

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()

gTime()

Returns a number, which is this time value. The number value is a millisecond representation of the specified Date object.

getTimezoneOffset()

gTimezoneOffset()

Returns (t * LocalTime(t)) / 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 a UTC Date object.

getUTCHours()

getUTCHours()

Returns HourFromTime(t).

getUTCMilliseconds()

getUTCMilliseconds()

Returns msFromTime(t).
getUTCMinutes()
getUTCMinutes()
Returns MinFromTime(t).

getUTCSseconds()
getUTCSseconds()
Returns SecFromTime(t).

getYear()
getYear()
Returns YearFromTime(LocalTime(t))—1900. The function getFullYear() is preferred for nearly all purposes because it avoids the year 2000 problem.

parse(string)
parsed(string)
Applies the ToString operator to its argument and interprets the resulting string as a date; it returns a number, the number which is a 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])

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 ])

Sets the [Value] property of this value to UTC time. Returns the value of the [Value] property of this value.

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)

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]])

Sets the [Value] property of this value to ECMAScript.Date. Returns the value of the [Value] property of this value.

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)

Sets the [Value] property of this value to time and returns the value of the [Value] property of this value.

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 322
- “length” on page 322
- “toString()” on page 322

Function(p1, p2, . . . , pn, body)

Function Constructor. The last argument specifies the body (executable code) of a function; any preceding arguments specify formal parameters.

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 322
- “eval(x)” on page 323
- “Infinity” on page 323
- “isFinite(number)” on page 323
- “isNaN( value )” on page 323
- “NaN” on page 323
- “parseFloat(string)” on page 323
- “parseInt(string, radix)” on page 323
- “unescape(string)” on page 323

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 (Not a Number), +*, 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 323) 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 324
- “LN10” on page 324
- “LN2” on page 325
- “LOG2E” on page 325
- “LOG10E” on page 325
- “PI” on page 325
- “SQRT1.2” on page 325
- “SQRT2” on page 325
- “abs(x)” on page 325
- “acos(x)” on page 325
- “asin(x)” on page 325
- “atan(x)” on page 326
- “atan2(x,y)” on page 326
- “ceil(x)” on page 326
- “cos(x)” on page 326
- “exp(x)” on page 326
- “floor(x)” on page 327
- “log(x)” on page 327
- “max(x,y)” on page 327
- “min(x,y)” on page 327
- “pow(x,y)” on page 327
- “random()” on page 327
- “round(x)” on page 328
- “sin(x)” on page 328
- “sqrt(x)” on page 328
- “tan(x)” on page 328

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 *, 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

cos(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 -infinity) 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.

dexp(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 +infinity) 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.

csqrt(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 329
- “MIN_VALUE” on page 329
- “NaN” on page 329
- “NEGATIVE_INFINITY” on page 329
- “Number()” on page 329
- “POSITIVE_INFINITY” on page 329
- “toString(radix)” on page 329
- “valueOf()” on page 329
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 330
- “toString()” on page 330
- “valueOf()” on page 330

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 331
- “charAt(pos)” on page 331
- “charCodeAt(pos)” on page 331
- “fromCharCode(char0, char1, . . .)” on page 331
- “indexOf(searchString, pos)” on page 331
- “lastIndexOf(searchString, pos)” on page 331
- “length” on page 331
- “match(RegExp)” on page 332
- “replace(RegExp, String)” on page 332
- “search(RegExp)” on page 332
- “split(separator)” on page 332
- “substring(start, end)” on page 332
- “toLowerCase()” on page 332
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()

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” on page 333
- “getEnvironmentLanguage” on page 333

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” on page 333
- “get()” on page 334
- “execService()” on page 334

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).

**Example:**
```
if ( IDVault.DNcompare(flowdata.get('Activity3/CardRequest/Candidate'),recipient )) true; else false ;
```

**get()**
```
get()
get(fieldname)
get(fieldname,dn)
get(fieldname,dn,entity-type)
get(fieldname,dn,entity-type,attribute)
```

This corresponds to the IDVault.get() function of the workflow script engine. Retrieves the value(s) of the attribute for the given entity. The result result is an array of values. If the field parameter is specified the result of the query result is used to refresh the content of the field. If not, result is up to the form developer to use the result of the query. Example:
```
IDVault.get("assetProp",dn,"user","LastName");
```

**execService()**
```
IDVault.execService(service)
IDVault.execService(service, param)
IDVault.execService(service, param, locale)
```

Executes an AJAX service, the result will used to refresh the content of the field. The service must be registered in the UI control registry. The first column of the result list result is used for the display value, the second one for the data value. Example:
```
var r=IDVault.execService("dnlookup2",params);
var res=r?r["_data"][_raw][dn]["value"]="error encountered";
field.setValues("IDVault.execService("dnlookup2") :"+res);
```
The Requests & Approvals tab in the Identity Manager User Application includes a group of actions called My Team’s Work. The My Team’s Work actions allow you to work with team member tasks and requests in a workflow. This section describes how to create a team and define its characteristics (such as members, manager, and request rights). Topics include:

- Section 11.1, “About Teams,” on page 335
- Section 11.2, “Managing Provisioning Teams,” on page 337

### 11.1 About Teams

A team identifies a group of users and determines who can manage provisioning requests and approval tasks associated with this team. The team definition consists of a list of team managers, team members, and team options, as described below:

- The **team managers** are those users who can administer requests and tasks for the team. Team managers can also be given permission to set proxies and delegates for team members. Team managers can be users or groups.
- The **team members** are those users who are allowed to participate on the team. Team members can be users, groups, or containers within the directory. Alternatively, they can be derived through directory relationships. For example, the list of members could be derived by the manager-employee relationship within the organization. In this case, the team members would be all users that report to the team manager.

**NOTE:** The Provisioning Application Administrator can configure the directory abstraction layer to support cascading relationships so that multiple levels within an organization can be included within a team. The number of levels to include is configurable by the administrator.

- The **team options** determine the provisioning request scope, which specifies whether the team managers can act on an individual provisioning request, one or more categories of requests, or all requests. The team options also determine whether team managers can set proxies for team members or set the availability of team members for the purpose of delegation.

**NOTE:** The User Application only supports a single level for proxy assignments. Proxy assignments are not propagated to multiple levels.

The Provisioning Application Administrator can perform all team management functions.

**Differences between teams and groups** Although a team can sometimes refer to a group in the Identity Vault, a team is not the same thing as a group. When you define a group in the Identity Vault, you identify a set of users that have something in common. However, the group does not automatically have the capabilities of a team within the User Application. To take advantage of the team capabilities within the User Application, you must define a team that points to the group.

- Section 11.1.1, “About Provisioning Team Requests,” on page 336
- Section 11.1.2, “Using a Team to Manage Direct Reports,” on page 336
11.1.1 About Provisioning Team Requests

A team request object specifies the requests that a provisioning team can work on. The request rights specify the actions that team managers can perform on the provisioning requests and tasks.

The team definition has a one-to-many relationship with team request objects. This means that each team must have at least one team request object defined for it, but it can have more. Each team request object is associated with only one team definition. When you configure the team request object, you configure the task scope and permissions for the team manager.

The task scope options define the manager’s ability to act on tasks:

- Where a team member is an addressee
- Where a team member is a recipient

**WARNING:** For security reasons, the recipient task scope option is disabled by default. Giving a team manager the ability to act on tasks where the recipient of the request is a team member can raise several security issues. First, the manager is then able to view data included on any of the forms that are displayed during the course of workflow execution, regardless of his or her trustee rights. Second, depending on the permission options (see below), a team manager could circumvent the approval process by claiming or approving the task, or by reassigning it to someone else.

The permissions options define the team manager’s ability to:

- Initiate a provisioning request on behalf of a team member.
- Retract a provisioning request on behalf of a team member.
- Make a team member a delegate for other team members’ provisioning requests.
- Claim a task for a team member who is a recipient or addressee (based on the task scope).
- Reassign a task for a team member who is a recipient or addressee (based on the task scope).

If both of the task scope options are disabled, the team manager cannot view or act on any active requests. Therefore, you must enable at least one of the Permissions options for the team manager.

**NOTE:** The User Application only supports a single level for delegate assignments. Delegate assignments are not propagated to multiple levels.

The trustee rights defined for a provisioning request apply to team managers who want to initiate a request on behalf of their team members. If the team manager does not have the trustee rights to a provisioning request definition, the team manager cannot make the request because the User Application does not display the provisioning request.

11.1.2 Using a Team to Manage Direct Reports

You can define a team that allows managers throughout an organization to control the provisioning environment for their direct reports. If defined properly, a single team definition can be used to allow all managers to control the activities of their direct reports. This means that you do not need to define a separate team for each reporting relationship.

Here are the basic requirements for a team that supports direct reports within an organization:

- The members of the team are defined by the Manager-Employee relationship.
The managers of the team are defined by a dynamic group that searches subcontainers, using a search filter that retrieves only the managers.

After the team has been defined, the User Application allows all managers to use the team management actions within the navigation menu. This gives the managers the ability to control the provisioning activities that their direct reports can perform.

For details on how to define a team to manage direct reports, see Section 11.2.3, “Creating a Team to Manage Direct Reports,” on page 343.

11.2 Managing Provisioning Teams

This section includes information about the following topics:

- Section 11.2.1, “Creating a Provisioning Team,” on page 337
- Section 11.2.2, “Deleting a Provisioning Team,” on page 343
- Section 11.2.3, “Creating a Team to Manage Direct Reports,” on page 343

11.2.1 Creating a Provisioning Team

To create a new provisioning team:

1. Launch the New Teams Wizard from the Provisioning view by right-clicking the Teams node, then selecting New. The New Provisioning Teams dialog box displays.

   ![New Provisioning Team dialog box]

2. Fill in the fields as follows:
3 Click *Finish*. The Team panel of the Provisioning Team editor displays.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>Type a common name (CN) for the team.</td>
</tr>
<tr>
<td>Display Label</td>
<td>Type the name of the provisioning team. This is the name displayed in Designer and also in the User Application runtime. The label is localizable in the Provisioning Team editor.</td>
</tr>
</tbody>
</table>

4 Type a description.

5 To add a Team Manager, click ![add icon], then choose the objects from the Identity Vault.
   You might be prompted for credentials before you are able to access the Identity Vault.
   Managers can be users or groups.
6 Select *Managers are members of team* if you want the manager to be included as a team member in requests submitted for the team.

7 To define the team’s members, do one of the following:
   - Click *DAL Relationship*, then select the relationship that represents the team’s membership.
   - Click *Identity Vault Objects*. Click , then select the members from the Identity Vault.

8 Complete the Options section of the panel as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Allow Manager(s) to set the availability of team members</em></td>
<td>When this setting is enabled, the team managers can access the <em>Team Availability</em> action in the navigation menu of the User Application.</td>
</tr>
<tr>
<td><em>Allow Manager(s) to set proxy for team members</em></td>
<td>When this setting is enabled, the team managers can access the <em>Team Proxy Assignments</em> action in the navigation menu of the User Application.</td>
</tr>
<tr>
<td><em>All team members will be displayed in a select list</em></td>
<td>When this option is selected, the manager can select team members in a drop-down list box. Use this option when the team has only a few members.</td>
</tr>
<tr>
<td><em>The manager will need to search for the member using a select-pick list</em></td>
<td>When this option is selected, the manager must perform a search before selecting team members. Use this option when the team has a large number of members.</td>
</tr>
</tbody>
</table>

If a particular team definition does not permit team managers to set proxies or team availability settings, the manager can still view the settings defined for the team members by the administrator or by a manager of another team to which these users belong. However, the team manager cannot edit these settings, view details for these settings, or create new proxy assignments or team availability settings.

You can now specify what types of requests this team can work on.
9 Click Requests. The Provisioning Team Request page displays.

10 Choose the Provisioning Request Scope for this team. The values are:

<table>
<thead>
<tr>
<th>Selection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Specifies that this team definition applies to all request types.</td>
</tr>
<tr>
<td>Categories</td>
<td>Specifies that this team definition applies to all request types associated with a particular category.</td>
</tr>
<tr>
<td>Individual Requests</td>
<td>Specifies that this team definition applies to a single request type.</td>
</tr>
</tbody>
</table>

11 Click and specify an identifier for the new provisioning team request object.
Specify values for the General panel. It displays different options, depending on the scope you select.

- For Categories, select one or more categories from the *Available Categories* column and move them to the *Selected Categories* column.
- For Individual requests, select the request from the drop-down list.

Each category or individual request can only be associated with one request object. As you select them, they are removed from the list of possible values for the next request object that you create.

Complete the Task Scope tab as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| Allow managers to act on tasks where a team member is an addressee | When this setting is enabled, the team managers can use the *Team Tasks* action within the User Application to take actions on tasks for which the team members are addressees. These actions include approving and denying requests.  
If you do not permit team managers to act on tasks for which the team member is an addressee, you can view these tasks, but you cannot see details about them, or take actions on them. |
| Allow managers to act on tasks where a team member is a recipient   | When this setting is enabled, the team managers can use the *Team Tasks* action within the User Application to take actions on tasks for which the team members are recipients. These actions include approving and denying requests.  
If you do not permit team managers to act on tasks for which the team member is a recipient, you can view these tasks, but you cannot see details about them, or take actions on them. |

**NOTE:** For security reasons, the recipient task scope option is disabled by default. Giving a team manager the ability to act on tasks where the recipient of the request is a team member can raise several security issues. First, the manager is then able to view data included on any of the forms that are displayed during the course of workflow execution, regardless of his or her trustee rights. Second, depending on the permission options, a team manager could circumvent the approval process by claiming or approving the task or reassigning it to someone else.

Complete the Permissions page as follows:
Click Save.

You must deploy the Provisioning Team for it to be available to the User Application. See Section 2.7, “Deploying Provisioning Objects,” on page 28. A deploy of a provisioning team creates two objects in the User Application driver Appconfig Teams node:

- srvprvTeam: contains the provisioning teams object.
- srvprvTeamRequest: contains the request object.

Because there are two objects stored in the User Application driver for a provisioning team, the compare and import operations are different than for other types of provisioning objects. To compare or import both objects, you must do the compare or import at the team node and not at the individual element. If you do the compare or import at the individual element, only the srvprvTeam object is imported or compared.
11.2.2 Deleting a Provisioning Team

You delete the Provisioning Team object from the Provisioning view by selecting the team, right-clicking, then selecting Delete. The Delete confirmation dialog box lets you specify whether to delete the object locally only, or from the Identity Vault during the next deploy of the parent object. If you delete team request objects in Designer, the team request objects are deleted from the Identity Vault when you deploy the team.

11.2.3 Creating a Team to Manage Direct Reports

1 In iManager, create a dynamic group called Managers.
   1a Set the Search Scope to Search Sub Containers.
   1b Specify the Search Filter as (&(isManager=TRUE)).
      For complete details on creating dynamic groups, see the Novell Identity Manager: Administration Guide.

2 In Designer, create a new provisioning team and name it DirectReports.
   2a To specify the team managers, pick the Managers dynamic group you created earlier.
   2b To identify the team members, select the Manager-Employee relationship.

   2c To define the team options:
      • Select Allow managers to set team availability for team members.
      • Select Allow managers to set proxies for team members.
      • Select All team members will display in a select list.

3 Select the Requests tab to define a provisioning team request object.

4 Set the Provisioning Request Scope to All Provisioning Requests, then name the new provisioning team request DirectReportsTeamRequestRights.

5 Click the Task Scope tab and specify the task scope options as follows:
   • Select Allow managers to act on tasks where the team member is an addressee.
   • Deselect Allow managers to act on tasks where the team member is a recipient.

6 Click the Permissions tab and specify the permission options as follows:
   • Select Allow managers to initiate a Provisioning Request on behalf of a team member.
   • Select Allow managers to retract a Provisioning Request on behalf of a team member.
   • Select Allow managers to make a team member a delegate for other team member’s Provisioning Requests.
   • Select Allow managers to claim a task for team members who are a recipient and/or addressee based on the task scope.
   • Select Allow managers to reassign a task for team members who are a recipient and/or addressee based on the task scope.

7 Save and deploy the team.

For more information, see Section 2.7, “Deploying Provisioning Objects,” on page 28.