User Application: Migration Guide

Novell® Identity Manager Roles Based Provisioning Module

3.7

December 17, 2009

www.novell.com

Legal Notices

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the latest online documentation for this and other Novell products, see the Novell Documentation Web page (http://www.novell.com/documentation).

Contents

About This Guide

This guide describes how to migrate to the Roles Based Provisioning Module 3.7 from an earlier version of the User Application or Roles Based Provisioning Module.

Audience

This guide is intended for administrators who are responsible for installing and maintaining 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.

Migrating to the Roles Based Provisioning Module Version 3.7

1

This section provides an overview of the steps for migrating to the Roles Based Provisioning Module 3.7 from an earlier version of the User Application or Roles Based Provisioning Module.

Topics include:

NOTE: This migration document frequently references installation steps found in the Roles Based Provisioning Module 3.7 Installation Guide (http://www.novell.com/documentation/idmrbpm37/index.html).

1.1 Migration Checklist

To migrate to the Novell Identity Manager Roles Based Provisioning Module 3.7, you must perform the following tasks:

- ❑ Before beginning the migration process, make directory and database backups of your Identity Manager system.
- ❑ Verify that your software meets the system requirements for the Roles Based Provisioning Module 3.7.

  See Section 1.3, "System Requirements" in the Roles Based Provisioning Module 3.7 Installation Guide (http://www.novell.com/documentation/idmrbpm37/install/data/bcyzv6p.html).
- ❑ Download the Identity Manager Roles Based Provisioning Module.

  See Section 2.2, "Downloading the Roles Based Provisioning Module" in the Roles Based Provisioning Module 3.7 Installation Guide (http://www.novell.com/documentation/idmrbpm37/install/data/i1070359.html).
- ❑ Check the following supporting components:
  - Make sure you have a supported Identity Manager Metadirectory installed.

    See Section 2.1, "Installing the Identity Manager Metadirectory" in the Roles Based Provisioning Module 3.7 Installation Guide (http://www.novell.com/documentation/idmrbpm37/install/data/bcz3a0b.html).
  - Make sure you have a supported application server installed.

    See Section 2.3, "Installing an Application Server" in the Roles Based Provisioning Module 3.7 Installation Guide (http://www.novell.com/documentation/idmrbpm37/install/data/bf7ojtd.html).

- Make sure you have a supported database installed.

  See Section 2.4, "Installing a Database" in the Roles Based Provisioning Module 3.7 Installation Guide (http://www.novell.com/documentation/idmrbpm37/install/data/bcz3524.html).

- ❑ If you are migrating from a 3.6 or 3.6.1 User Application, perform the pre-migration tasks for migrating the security model.

  See Section 1.4, "Migrating the Security Model," on page 16.

- ❑ Update the Roles Based Provisioning Module Metadirectory components.

  See Section 3.0, "Installing the Roles Based Provisioning Module on the Metadirectory" in the Roles Based Provisioning Module 3.7 Installation Guide (http://www.novell.com/documentation/idmrbpm37/install/data/rbpminstall.html).

  IMPORTANT: Since you are migrating from a previous version of the RBPM, it is mandatory that you rune the NrfCaseUpdate utility, as instructed in this utility.

- ❑ Migrate the User Application driver in Designer for Identity Manager 3.5. See Section 1.2, "Migrating User Application Drivers," on page 10.

  IMPORTANT: You must use Designer for Identity Manager 3.5. You cannot use iManager. For details on downloading Designer for Identity Manager 3.5, see the Novell download page (http://download.novell.com/index.jsp).

- ❑ Update the Role and Resource Service Driver in iManager or Designer for Identity Manager 3.5.
  - For iManager, see Section 4.2, "Creating the Role and Resource Driver in iManager" in the Roles Based Provisioning Module 3.7 Installation Guide (http://www.novell.com/documentation/idmrbpm37/install/data/bcvelpn.html).
  - For Designer, see the Identity Manager User Application: Design Guide (http://www.novell.com/documentation/idmrbpm37).

  IMPORTANT: If you have an existing Role Service driver for version 3.6 or 3.6.1, you must delete the existing driver and create a new one for version 3.7. Remember to start your new Role and Resource Service driver once it is created.

- ❑ Update the Identity Manager User Application.

  See Section 1.3, "Migrating the User Application," on page 13.

- ❑ If you are migrating from a 3.6 or 3.6.1 User Application, perform the post-migration tasks for migrating the security model.

  See Section 1.4.3, "Steps for Migrating the System Roles and Team Permissions," on page 23.

1.2 Migrating User Application Drivers

- Section 1.2.1, "About User Application Driver Migration," on page 11
- Section 1.2.2, "Updating Default Notification Templates," on page 11
- Section 1.2.3, "Migrating a User Application Driver," on page 12
- Section 1.2.4, "Deploying the Migrated Driver," on page 13

Role Service drivers are not migrated. If you have an existing Role Service driver for version 3.6 or 3.6.1, you must create a new one for version 3.7.

1.2.1 About User Application Driver Migration

Migrating the User Application driver ensures that the driver contains the necessary configuration for new features, maintains support for existing configurations (for example, custom entities and provisioning request definitions), adds new driver properties, and updates the following:

- Directory abstraction layer definitions
- Provisioning request definitions
- Roles and resources
- Several non-visible runtime configuration objects

You can set preferences for migrating the User Application driver (see the section on setting provisioning view preferences in the *Identity Manager User Application: Design Guide (http://www.novell.com/documentation/idmrbpm37/index.html)*.

The User Application driver migration utility migrates only the User Application configuration of the User Application driver. It does not migrate the entire driver. Specifically, it doesn't migrate policies. These are handled by the general Designer and the deploy code.

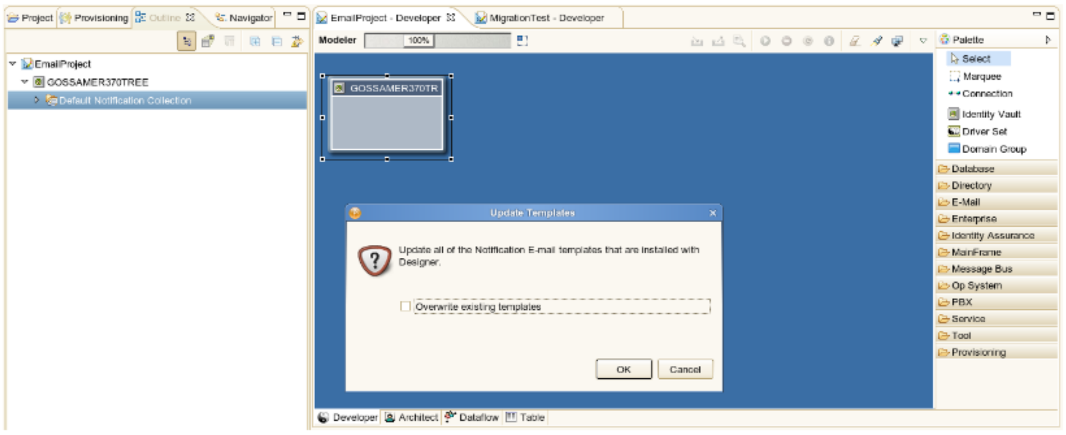
1.2.2 Updating Default Notification Templates

The latest Default Notification Templates do not get automatically deployed to your Identity Vault during migration. These templates are required for all role, resource, and attestation assignments to work. You will need to manually add and deploy these required templates before migrating your User Application driver. Otherwise, your User Application driver migration will end in warnings.

Use the following steps to deploy the notification templates:

1. Launch Designer 3.5 and create a Project.
2. In Outline view, right-click on the Project name and select *Live>Import*. Select *Default Notification Collection* from eDirectory.
3. Right-click the *Default Notification Collection* and select *Add All Templates*.

   If you have made changes to any of the default notification templates, make sure to uncheck the *Overwrite existing templates* checkbox so that your changes do not get overwritten.



Update all of the Notification E-mail templates that are installed with Designer.

Overwrite existing templates

4 Click *OK*.

5 Right-click the *Default Notification Collection* again and choose *Live>Deploy* to deploy all the templates.

1.2.3 Migrating a User Application Driver

1 Make a backup copy of the provisioning project that contains the User Application driver to migrate:

1a Right-click the name of the project in Project view, then select *Copy Project*.

1b In the *Copy Project* dialog box, type a new *Project Name* (or accept the default name), then click *OK*.

2 Use one of the following methods to run the Migrate command for the User Application that you want to migrate.

- In the *Provisioning View*, right-click the name of the User Application driver and select *Migrate*.
- In the *Modeler* view, right-click the name of the User Application driver and select *Application > Migrate*.
- In the *Outline* view, right-click the name of the User Application driver, and select *Migrate*.

This migrates the local definition of the driver found in the Designer workspace. These changes are not made to the Identity Vault until after you deploy the changes.

3 If the *Close All Editors* dialog box displays, select *Yes*.

If you have unsaved work in an open editor, the *Save Resource* dialog box displays.

4 If the *Save Resource* dialog box is displayed, select *Yes* to save changes in open editors.

5 If you are prompted for a driver version to migrate to, select *IDM 3.7 (Roles Based Provisioning Module)*, then click *OK*.

6 Designer warns you that schema changes are required in the Identity Vault. Select *Yes* to continue the migration (if you have already updated the Identity Vault schema in a separate process), or select *No* to cancel the migration (if you have not already updated the Identity Vault schema).

NOTE: The Identity Vault schema is updated when you run the RBPM Installation.

When the migration completes, Designer displays a dialog box listing information, warnings, and errors encountered during migration. For example, for all objects added during migration, informational text displays for those objects. You can perform several operations by using this dialog box.

- The migration does not automatically save a log file describing the content of the dialog box to the `project-name/Provisioning/AppConfig` folder for the associated User Application. You can manually save it here. The default name of the file is `migrationLogdate.log`. You can also save the migration log to a file anywhere on disk by clicking the floppy disk icon in the upper right corner of the dialog box.
- To revert to the original User Application configuration (for example, if errors occurred during the migration) click the *Undo Migration* button.

  You can undo the migration until you click *OK* in the dialog box.
- To deploy the driver, you can select *Deploy migrated User Application*; or, if you want to deploy the driver later after validating the project using the project checker, follow the steps in Deploying the Migrated Driver to complete the process.

NOTE: When you migrate a driver, ensure that all other drivers in the same driver set are also migrated to the same version.

1.2.4 Deploying the Migrated Driver

The driver migration is not complete until you deploy the entire driver to the Identity Vault.

1. Open the project in Designer and run the Project Checker on the migrated objects.

   See *"Validating Provisioning Objects"* in the Identity Manager User Application: Design Guide (http://www.novell.com/documentation/idmrbpm37/index.html). If validation errors exist for the configuration, you are informed of the errors. These errors must be corrected before you can deploy the driver.
2. Navigate to the Outline view, right-click the driver, and select *Deploy*, or in the Modeler view, right-click the driver and choose *Driver* > *Deploy*.

   After the migration, the project is in a state in which only the entire migrated configuration can be deployed. You cannot import any definitions into the migrated configuration. After the entire migration configuration has been deployed, this restriction is lifted, and you can deploy individual objects and import definitions.
3. Repeat this process for each User Application driver in the driver set.

1.3 Migrating the User Application

This section provides an overview of the process for migrating the User Application. The installation steps described below are required for migration.

1.3.1 Installing the User Application

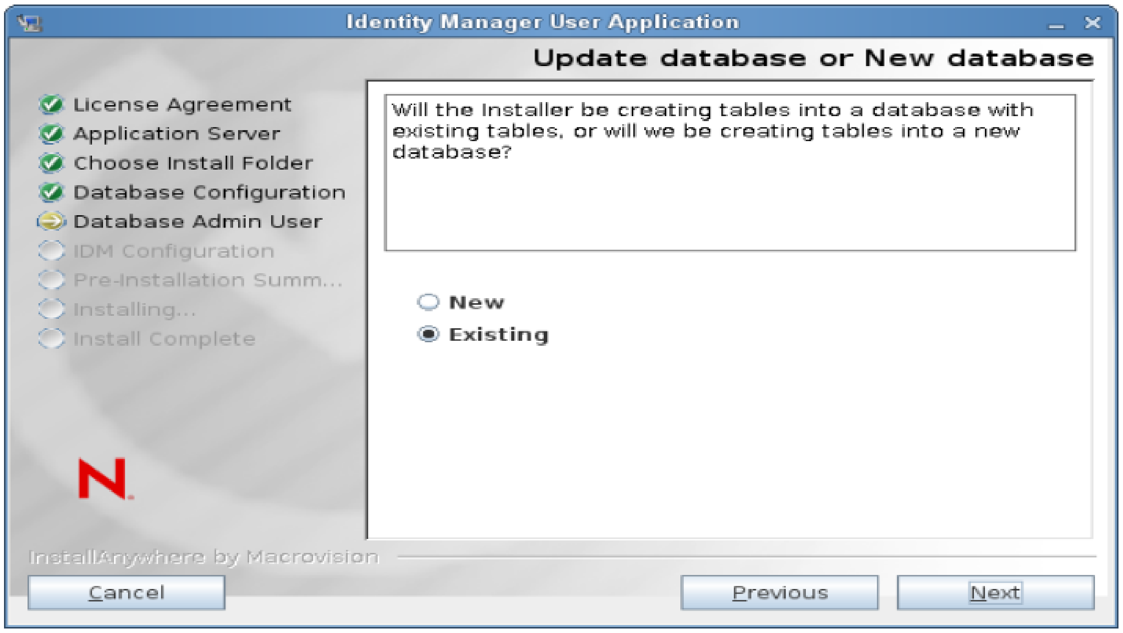
IMPORTANT: You must have the correct JDK installed before you start the installation program. See Section 2.5, "Installing the Java Development Kit" in the Install Guide (http://www.novell.com/documentation/idmrbpm37/install/data/bcz3432.html).

You can migrate your User Application by launching the installation program in one of three modes:

- Graphical user interface. Depending on which application server you're running, you should look at one of the following sections:
  - Section 5.0, "Installing the User Application on JBoss" (http://www.novell.com/documentation/idmrbpm37/install/data/bcz57yw.html)
  - Section 6.0, "Installing the User Application on WebSphere" (http://www.novell.com/documentation/idmrbpm37/install/data/bcz598n.html)
  - Section 7.0, "Installing the User Application on WebLogic" (http://www.novell.com/documentation/idmrbpm37/install/data/bf53fpm.html)
- Console (command line) interface. See Section 8.1, "Installing the User Application from the Console" (http://www.novell.com/documentation/idmrbpm37/install/data/b70zagk.html).
- Silent install. See Section 8.2, "Installing the User Application with a Single Command" (http://www.novell.com/documentation/idmrbpm37/install/data/b8m646i.html)

When you run the User Application installation program, you must use the same User Application database that you used for the previous installation (that is, the installation from which you are migrating). You can use a different User Application context name.

During the installation program, when asked if you want to update your database data from a previous installation, make sure that *Existing* is selected.



Update database or New database

Will the Installer be creating tables into a database with existing tables, or will we be creating tables into a new database?

New [✓ Existing]

1.3.2 Post-Installation Steps

This section provides information about User Application migration steps required after installation. Topics include:

- "Updating the SharedPagePortlet Maximum Timeout Setting" on page 15
- "Turning Off the Automatic Query Setting for Groups" on page 15

Updating the SharedPagePortlet Maximum Timeout Setting

The SharedPagePortlet's entry in the portlet.xml file within the WEB-INF directory has been changed from:

```
<expiration-cache>-1<expiration-cache>
```

to:

```
<expiration-cache>0<expiration-cache>
```

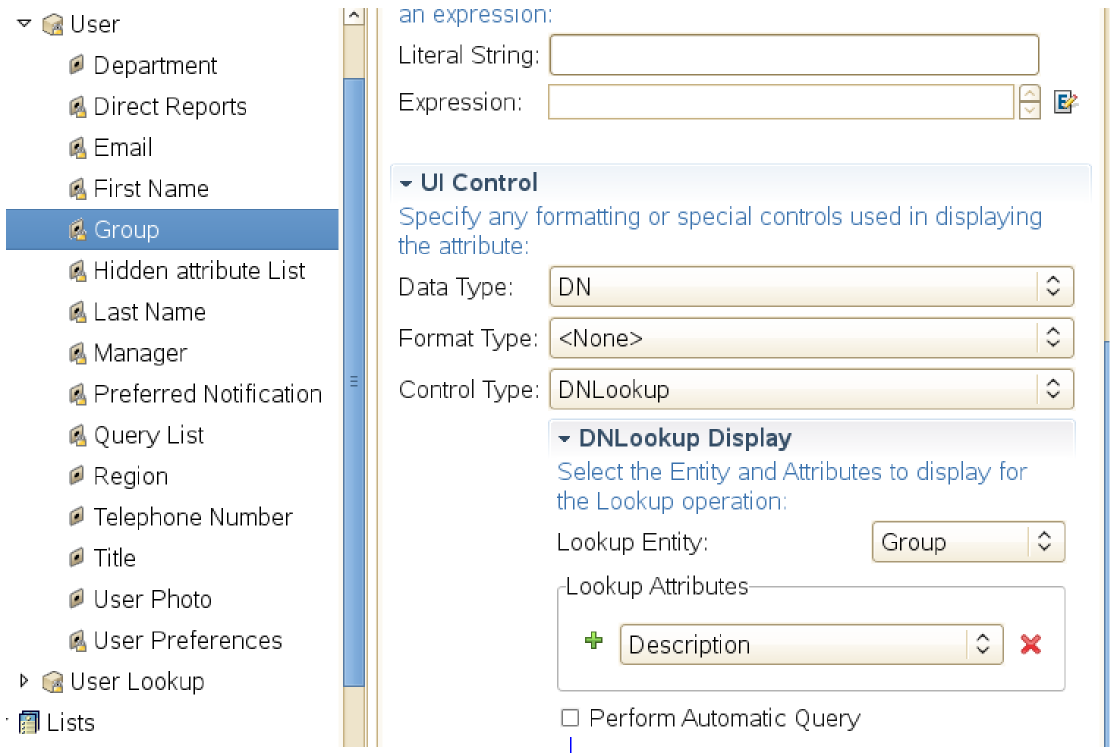
If you have customized any of the default settings or preferences for the SharedPagePortlet, then it has been saved to your database and this setting will get overwritten. As a result, navigating to the Identity Self-Service tab may not always highlight the correct Shared Page. To be sure that you do not have this problem, follow these steps:

1. Login as a User Application Administrator.
2. Navigate to *Administration* > *Portlet Administration*.
3. Expand *Shared Page Navigation* and click on *Shared Page Navigation* in the portlet tree on the left hand side of the page.
4. Click on the *Settings* tab on the right hand side of the page.
5. If *Maximum Timeout* is not set to 0, set it to 0, and click *Save Settings*.

Turning Off the Automatic Query Setting for Groups

By default, the DNLookup Display for the Group entity in the Directory Abstraction Layer is turned on. This means that whenever the object selector is opened for a group assignment, all the groups are displayed by default without the need to search them. You should change this setting, since the window to search for groups should be displayed without any results until the user provides input for search.

You can change this setting in Designer by simply unchecking the Perform Automatic Query checkbox, as shown below:



an expression:

Literal String:

Expression:

UI Control

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

Data Type: DN

Format Type: <None>

Control Type: DNLookup

DNLookup Display

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

Lookup Entity: Group

Lookup Attributes

Description

☐ Perform Automatic Query

uncheck this if you don't want the autoquery to occur

1.4 Migrating the Security Model

This section describes the process of migrating the security model from RBPM 3.6 and 3.6.1 to the Roles Based Provisioning Module 3.7.

- Section 1.4.1, "Security Model Overview," on page 16
- Section 1.4.2, "Pre-Migration Steps," on page 17
- Section 1.4.3, "Steps for Migrating the System Roles and Team Permissions," on page 23
- Section 1.4.4, "Sample Code," on page 32

1.4.1 Security Model Overview

The RBPM 3.7 security model supports three levels of administration:

- Domain Administrators
- Delegated Administrators (Domain Managers)
- Business line managers (Team Managers)

The security model exposes several authorization domains (feature areas):

- Roles
- Resources

- Provisioning
- Compliance
- RBPM Configuration
- RBPM Security

Delegated administration is supported across three of these domains:

- Roles
- Resources
- Provisioning

Permissions for delegated administrators and business line administrators are governed by eDirectory access control lists.

For more information on the new security model, see the Identity Manager User Application: Administration Guide (http://www.novell.com/documentation/idmrbpm37/index.html).

1.4.2 Pre-Migration Steps

Before you migrate, you need to:

1. Record existing assignments for roles being deprecated in RBPM 3.7:
   - Provisioning Administrator
   - Attestation Manager
   - Role Manager
   - Security Officer
   - Auditor
2. Record all existing permissions for teams

The remainder of this section describes the steps for recording existing assignments for each of the system roles.

Provisioning Administrator

In previous versions of RBPM, the Provisioning Administrator was assigned using portal security. Now, it needs to be migrated to the new security model. RBPM 3.6.1 defines the Provisioning Module Administrator as a Portal Administrator assignment. The assignments are stored in the database. In 3.7, the Provisioning Administrator is defined by system role assignments. The initial assignment is specified during application installation and processed once for each User Application driver. The initialization is time stamped.

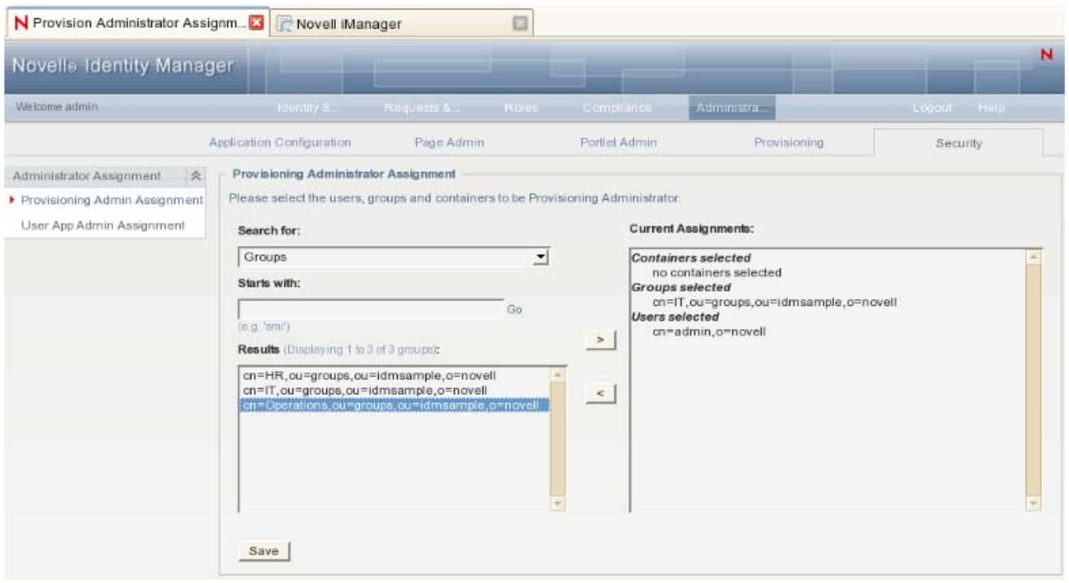
After initialization, additional assignments can be added or removed in the RBPM Administration section of the User Application.

You need to compile a list of identities that were assigned as Provisioning Administrator before proceeding with the upgrade by executing the following SQL statement:

```
select PRINCIPALNAME from securitypermissions where PERMISSIONIID like (select
IID from securityaccessrights where FWELEMENTIID like
'ProvisioningLocksmithElementId');
```

Alternatively, you can get the list from the Provisioning Administrator Configuration page in the User Application by performing the following steps:

1. Login to the User Application as administrator.
2. Navigate to *Administration->Security*.
3. Select *Provisioning Admin Administrator* in the left navigation menu and compile a list of all the identities (users, groups, and containers) currently assigned as Provisioning Administrator.



Provisioning Administrator Assignment

Please select the users, groups and containers to be Provisioning Administrator

Search for: Groups

Starts with:

(e.g. 'smi')

Results (Displaying 1 to 3 of 3 groups):

```
cn=HR,ou=groups,ou=idmsample,o=novell
cn=IT,ou=groups,ou=idmsample,o=novell
cn=Operations,ou=groups,ou=idmsample,o=novell
```

Current Assignments:

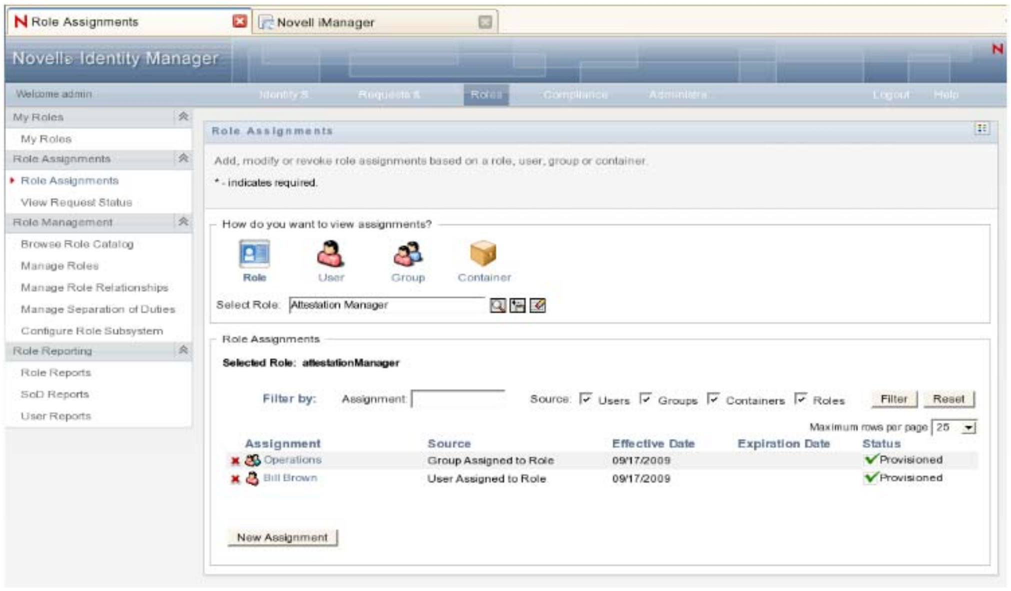
```
Containers selected
    no containers selected
Groups selected
    cn=IT,ou=groups,ou=idmsample,o=novell
Users selected
    cn=admin,o=novell
```

Attestation Manager

In RBPM 3.6.1, the Attestation Manager and Compliance Administrator system roles were available out of the box. Both of them gave assigned users an equal set of access rights within the Compliance domain. In RBPM 3.7, the Attestation Manager role is obsolete and assignments need to be migrated.

To determine the Attestation Manager assignments:

1. Compile a list of identities (users, groups, containers, and roles) that have Attestation Manager assigned by performing the following steps before migration:
   - **1a** Login to the User Application as a role administrator.
   - **1b** Navigate to the *Roles* tab.
   - **1c** Select *Role Assignments* in the left navigation menu.
   - **1d** Lookup the Attestation Manager role and compile a list of all identities assigned as Attestation Manager (users, groups, containers). Let's call the list attestation manager assignment list.



Role Assignments

Add, modify or revoke role assignments based on a role, user, group or container.

* - indicates required.

How do you want to view assignments?

Select Role: Attestation Manager

Role Assignments

Selected Role: attestationManager

Filter by: Assignment: Source: [✓] Users [✓] Groups [✓] Containers [✓] Roles

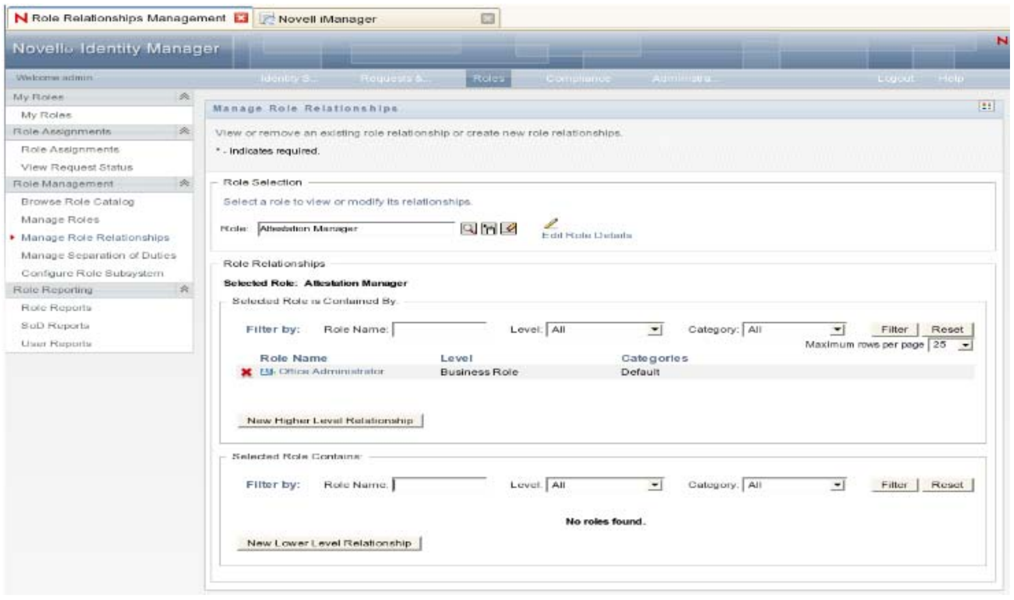
Maximum rows per page 25

Assignment	Source	Effective Date	Expiration Date	Status
Operations	Group Assigned to Role	09/17/2009		✓Provisioned
Bill Brown	User Assigned to Role	09/17/2009		✓Provisioned

1e Select *Manage Role Relationships* in the left navigation menu.

1f Lookup Attestation Manager role and add all parent role assignments to attestation manager assignment list.

1g Note if the Attestation Manager role has any child role assignments. If it does, compile the list of child roles.



Manage Role Relationships

View or remove an existing role relationship or create new role relationships.

* - indicates required.

Role Selection

Select a role to view or modify its relationships.

Role: Attestation Manager

Role Relationships

Selected Role: Attestation Manager

Selected Role is Contained By

Filter by: Role Name: Level: All Category: All

Maximum rows per page 25

Role Name	Level	Categories
Office Administrator	Business Role	Default

Selected Role Contains

Filter by: Role Name: Level: All Category: All

No roles found.

Role Manager

The RBPM 3.7 security model introduces the concept of delegated administration across the Provisioning, Roles, and Resources domains. In RBPM 3.6 and 3.6.1, delegated administration was supported for the Roles module only. As in RBPM 3.6 and RBPM 3.6.1, RBPM 3.7 delegated administrators are specified by system role assignments. The difference from previous versions is that delegated administrator permissions are not governed by browse rights access control lists. Instead, they are defined by a much more granular permission set based on the access attributes.

RBPM 3.6.1 defines three different roles: Role Manager, Security Officer, and Role Auditor. In RBPM 3.7, these three roles have been replaced with a single role called Role Manager. For this reason, role assignments and permissions need to be migrated to the RBPM 3.7 model.

To determine the Role Manager assignments:

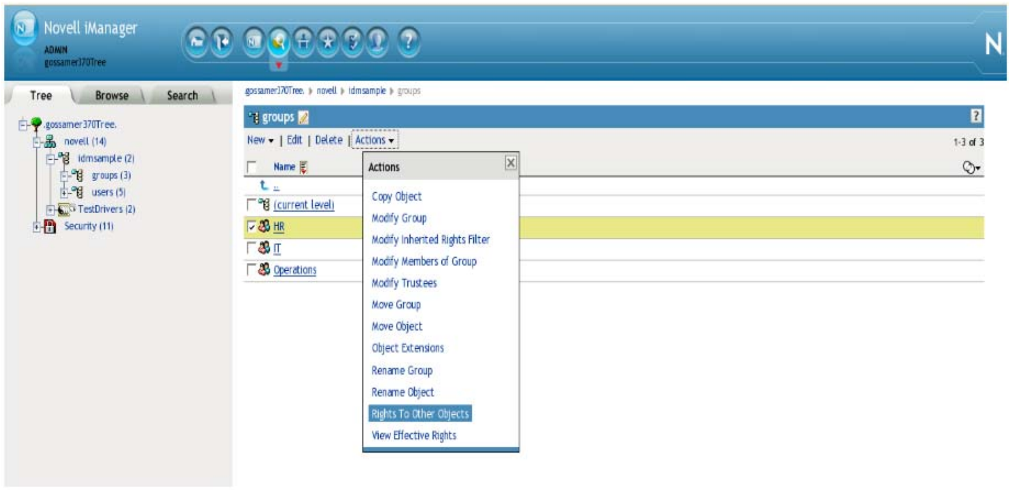
1 Compile a list of identities (users, groups, containers, and roles) that have Role Manager assigned by performing the following steps before migration:

- **1a** Login to the User Application as Role Administrator.
- **1b** Navigate to the *Roles* tab.
- **1c** Select *Role Assignments* in the left navigation menu.
- **1d** Lookup the Role Manager role and compile a list of all identities assigned as Role Manager (users, groups, containers). Let's call the list role manager assignment list.
- **1e** Select *Manage Role Relationships* in the left navigation menu.
- **1f** Lookup Role Manager role and add all parent role assignments to role manager assignment list.

2 Compile a list of roles for each identity, assigned as Role Manager, having browse rights for a user, group, container. This can be done via iManager, Designer, or an LDAP browser. Alternatively, you can develop a Java routine (as shown in Section 1.4.4, "Sample Code," on page 32).

To determine the Role browse rights for an identity in iManager:

- **2a** Go to *View Objects*.
- **2b** Browse the eDirectory Tree to your identity.
- **2c** Select the identity in the table in the right pane and select *Actions* > *Rights to Other Objects*.



2d In the *Rights to Other Objects* dialog that appears, check the *Search entire subtree* check box and press *OK*.

2e Make a note of all of the roles and role containers for which the identity has Browse rights.

Security Officer

To determine the Security Officer assignments:

1. Compile a list of the identities (users, groups, containers and roles) that have Security Officer assigned. This can be done by logging in as security administrator, going to the Role Catalog and viewing the list of assignments for the Security Officer role. Alternately, you can develop a Java utility (as described in Section 1.4.4, "Sample Code," on page 32). Here are the steps to compile the Security Officer assignment list before migration:

   1a Login to the User Application as Role Administrator.

   1b Navigate to the *Roles* tab.

   1c Select *Role Assignments* in the left navigation menu.

   1d Lookup the Security Officer role and compile a list of all identities assigned as Security Officer (users, groups, containers). Let's call the list security officer assignment list.

   1e Select *Manage Role Relationships* in the left navigation menu.

   1f Lookup the Security Officer role and add all parent role assignments to the security officer assignment list.

   1g If the Security Officer role has any child role assignments, compile the list of child roles.

2. Compile a list of SoDs for each identity, assigned as Security Officer, having browse rights to the user, group, or container. This can be done via iManager, Designer, or an LDAP browser. Alternatively, you can develop a Java routine (as described in Section 1.4.4, "Sample Code," on page 32).

   To determine the SoD browse rights for an identity in iManager:

   2a Go to *View Objects*.

   2b Browse the eDirectory Tree to your identity.

   2c Select the identity in the table in the right pane and select *Actions* > *Rights to Other Objects*.

   2d In the *Rights to Other Objects* dialog that appears, check the *Search entire subtree* check box and press *OK*.

   2e Make note of all the SoDs for which the identity has Browse rights.

Auditor

To determine the Auditor assignments:

1. Compile a list of identities (users, groups, containers, and roles) that have Auditor assigned. This can be done by logging in as security administrator, going to the Role Catalog and viewing the list of assignments for the Auditor role. Alternately, you can develop a Java utility (as described in Section 1.4.4, "Sample Code," on page 32).

   Here are the steps to compile the Auditor assignment list before migration:

   1a Login to the User Application as Role Administrator.

1b Navigate to the *Roles* tab.

1c Select *Role Assignments* in the left navigation menu.

1d Lookup the Auditor role and compile a list of all identities assigned as Auditor (users, groups, containers). Let's call the list auditor assignment list.

1e Select *Manage Role Relationships* in the left navigation menu.

1f Lookup the Auditor role and add all parent role assignments to the auditor assignment list.

1g If the Auditor role has any child role assignments, compile the list of child roles.

Team Permissions

In RBPM 3.6.1 and before, Novell supported teams for provisioning (workflow) administration only. Teams were defined as instances of the srvprvTeam object class. Permissions on the teams were defined by attributes on srvprvTeam instance and srvprvTeamRequest object attributes. In RBPM 3.7, teams are supported for Provisioning, Roles, and Resources delegated administration. Teams are defined as instances of the srvprvRbpmTeam class. Permissions are controlled by permission set based on the access attributes. The team is set as trustee for all ACLs. If you use provisioning teams, you need to migrate these teams to the new team model.

Migration is a two part process involving team definition migration and team permission migration, as described below:

- Team definition migration is automatically done for you in Designer 3.5 during User Application driver migration. For more information, see the Designer documentation on User Application driver migration.
- Team permission migration consists of pre and post migration steps. The pre-migration step is done by compiling list of team permissions before driver migration using Designer or iManager.

  To find team permissions in Designer, you need to record the permissions for each team. Open the team in the Team Editor, and record the permission settings for the following properties:

  - Allow manager(s) to set proxy for team members
  - Allow manager to set the availability of team members
  - Allow managers to initiate a Provisioning Request on behalf of a team member
  - Allow managers to make a team member a delegatee for other team member's Provisioning Requests
  - Allow managers to retract a Provisioning Request on behalf of a team member
  - Allow managers to reassign a task for team members who are a recipient based on the task scope
  - Allow managers to claim a task for team members who are an addressee based on the task scope

  To find team permissions in iManager:

  1. Go to View Objects.
  2. Browse the eDirectory tree until you find the TeamDefs container.
  3. For each team and team request, record the permission settings for the following attributes:
     - srvprvAllowMgrSetProxy

- srvprvAllowMgrSetAvailability
- srvprvAllowMgrInitiate
- srvprvAllowMgrSetDelegate
- srvprvAllowMgrRetract
- srvprvAllowMgrTaskReassign
- srvprvAllowMgrTaskClaim
- srvprvTaskScopeAddressee

1.4.3 Steps for Migrating the System Roles and Team Permissions

This section reviews the steps for migrating the system role assignments and team permissions. Topics include:

- "Migrating the Provisioning Administrator Assignments" on page 23
- "Migrating the Attestation Manager Assignments" on page 24
- "Migrating the Role Manager Assignments" on page 25
- "Migrating the Security Officer Assignments" on page 26
- "Migrating the Auditor Assignments" on page 27
- "Migrating the Team Permissions" on page 28

Migrating the Provisioning Administrator Assignments

After the User Application driver migration has been completed, assign one of the users to be Provisioning Administrator during the User Application installation time, and add the rest of the user, group, and container assignments using the RBPM Administration interface within the User Application.

Follow these steps to migrate the Provisioning Administrator assignments:

1 Login to the User Application as Security Administrator.

2 Navigate to *Administration->RBPM Provisioning and Security*.

3 Select *Administrator Assignments* in the left navigation menu.

4 Migrate the assigned identities by repeating the following steps for each identity in the list you compiled before the migration:

4a Select *Assign* to open the *New Administrator Assignment* dialog.

4b Select the *Provisioning* domain.

4c Select identity type (user, group container) and lookup the identity.

4d Check the *All Permissions* check box.

4e Click on the *Assign* button.

Migrating the Attestation Manager Assignments

After the User Application driver migration has been completed, add all identity (user, group, container, or role) assignments using the RBPM Administration interface within the User Application.

Follow these steps to migrate the Attestation Manager assignments:

1 Login to the User Application as Security Administrator.

2 Navigate to *Administration->RBPM Provisioning and Security*.

3 Select *Administrator Assignments* in the left navigation menu.

4 Migrate the assigned identities by repeating the following steps for each identity in the list you compiled before the migration:

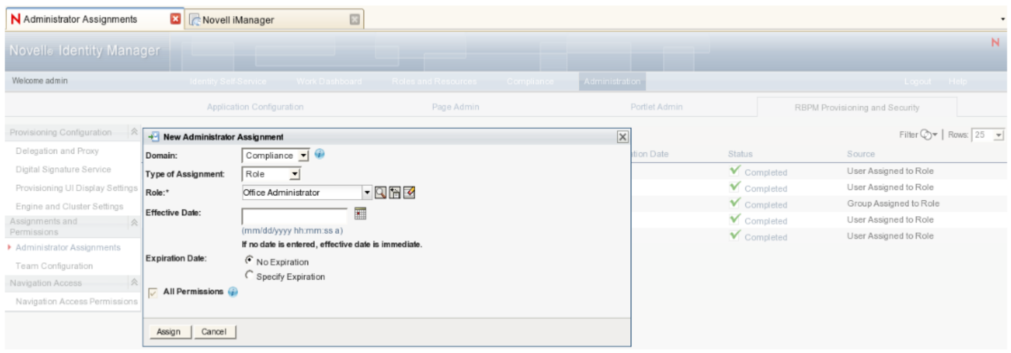
    4a Select *Assign* to open the *New Administrator Assignment* dialog.

    4b Select the *Compliance* domain.

    4c Select identity type (user, group container) and lookup the identity.

    4d Check the *All Permissions* check box.

    4e Click on the *Assign* button.


New Administrator Assignment

Domain: Compliance

Type of Assignment: Role

Role:* Office Administrator

Effective Date:

(mm/dd/yyyy hh:mm:ss a)

If no date is entered, effective date is immediate.

Expiration Date: [✓ No Expiration] Specify Expiration

[✓] All Permissions

Status	Source
✓ Completed	User Assigned to Role
✓ Completed	User Assigned to Role
✓ Completed	Group Assigned to Role
✓ Completed	User Assigned to Role
✓ Completed	User Assigned to Role

Administrator Assignments
Novell iManager
Novell® Identity Manager
Welcome admin
Identity Self-Service
Work Dashboard
Roles and Resources
Compliance
Administration
Logout
Help
Application Configuration
Page Admin
Portlet Admin
RBPM Provisioning and Security
Provisioning Configuration
Delegation and Proxy
Digital Signature Service
Provisioning UI Display Settings
Engine and Cluster Settings
Assignments and Permissions
Administrator Assignments
Team Configuration
Navigation Access
Navigation Access Permissions
Filter
Rows: 25
Assign
Cancel

    4f Process child roles list by creating a new non-system role, and assigning all roles from the child role list to this role. Assign all users, groups, containers and roles from the attestation manager list to the newly created role.

After the migration is done and verified, remove the obsolete system roles in the User Application:

1 Log in as a Security Administrator.

2 Go to *RBPM Administration>Administrator Assignments*.

3 Edit the assignment of the Security Administrator role to be Role Manager.

4 Add the *Delete Role* permission for the obsolete roles: Security Officer, Attestation Manager, and Auditor. For more information on assigning role permissions, see the Identity Manager User Application: Administration Guide (http://www.novell.com/documentation/idmrbpm37/index.html).

5 Click on the *Roles and Resources* tab.

6 Select *Role Catalog* from the left navigation menu.

7 Select the obsolete role from the list and click the *Delete* action. For more information on how to delete roles, see the Identity Manager User Application: User Guide (http://www.novell.com/documentation/idmrbpm37/index.html).

Migrating the Role Manager Assignments

After the User Application driver migration has been completed, you need to configure permissions for each identity from the identity list. Browse rights in 3.6.1 are equivalent to the Modify, Update, Delete, Assign to User, Revoke from User, Assign To Group and Container, Revoke from Group and Container permissions. The Browse right to a Role container is equivalent to the Create, Modify, Update, Delete, Assign to User, Revoke from User, Assign To Group & Container, and Revoke from Group & Container permissions.

Follow these steps to migrate the Role Manager assignments:

1. Login to the User Application as Security Administrator.
2. Navigate to *Administration->RBPM Provisioning and Security*.
3. Select *Administrator Assignments* in the left navigation menu.
4. Migrate the assigned identities by repeating the following steps for each identity in the list you compiled before the migration:

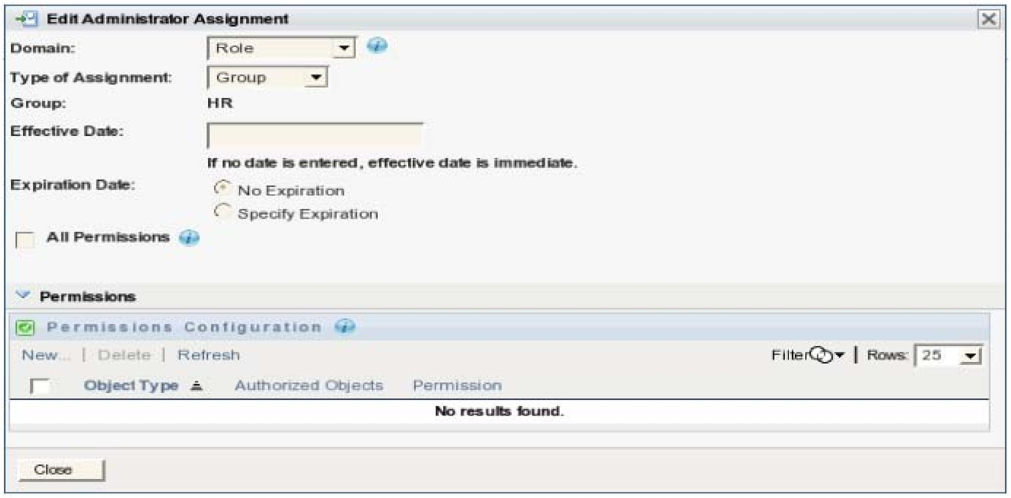
   4a Select *Assign* to open the *New Administrator Assignment* dialog.

   4b Select the *Role* domain.

   4c Select identity type (user, group container) and lookup the identity.

   4d Uncheck the *All Permissions* check box.

   4e Click *Assign*.



Edit Administrator Assignment

Domain: Role

Type of Assignment: Group

Group: HR

Effective Date:

If no date is entered, effective date is immediate.

Expiration Date: [✓ No Expiration] Specify Expiration

All Permissions

Permissions

Permissions Configuration

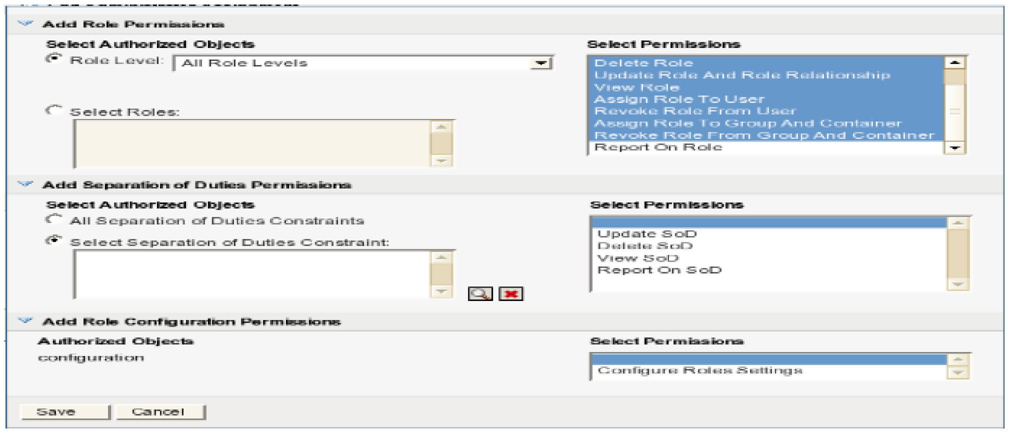
New... | Delete | Refresh Filter | Rows: 25

	Object Type	Authorized Objects	Permission
No results found.			

   4f Do not close the *New Administrator Assignment* dialog.

   4g In the *Permissions Configuration* section, select *New* to open the Permissions dialog.

   4h In the *Add Role Permissions* section, select all the role permissions except the Report on Role permission.



Add Role Permissions

Select Authorized Objects

(•) Role Level: All Role Levels

( ) Select Roles:

Select Permissions

Delete Role
Update Role And Role Relationship
View Role
Assign Role To User
Revoke Role From User
Assign Role To Group And Container
Revoke Role From Group And Container
Report On Role

Add Separation of Duties Permissions

Select Authorized Objects

( ) All Separation of Duties Constraints

(•) Select Separation of Duties Constraint:

Select Permissions

Update SoD
Delete SoD
View SoD
Report On SoD

Add Role Configuration Permissions

Authorized Objects

configuration

Select Permissions

Configure Roles Settings

Save Cancel

4i In the *Select Authorized Objects* section, lookup all the roles from the roles list compiled for the current identity.

4j Click *Save*.

4k Click *Close*.

Migrating the Security Officer Assignments

After the User Application driver migration has been completed, you need to assign each Security Officer identity from the list you compiled to be a Role Manager.

Follow these steps to migrate the Security Officer assignments:

1 Login to the User Application as a Security Administrator.

2 Navigate to *Administration>RBPM Provisioning and Security*.

3 Select *Administrator Assignments* in the left navigation menu.

4 Migrate assigned identities by repeating the following steps for each identity in the list you compiled before the migration:

4a Select *Assign* to open the *New Administrator Assignment* dialog.

4b Select the *Role* domain.

4c Select identity type (user, group container) and lookup the identity.

4d Uncheck the *All Permissions* check box.

4e Click *Assign*.

4f Do not close the *New Administrator Assignment* dialog.

Next, you need to configure the permissions for each user. The Browse right in 3.6.1 is equivalent to the Modify, Update, and Delete SoD permission. The Browse right to an SoD container is equivalent to the Create, Modify, Update, and Delete SoD permission.

4g In the *Permissions Configuration* section, select *New* to open the Permissions dialog.

4h In the *Add Separation of Duties Permissions* section, select all SoD permissions except the Report On SoD permission.

4i Lookup all SoDs from the SoDs list compiled for current identity.

4j Click *Save* button.

4k Close the *New Administrator Assignment* dialog, by clicking on *Close* button.

4l Process child roles list by creating a new non-system role, and assigning all roles from the child role list to this role. Assign all users, groups, containers, and roles from the attestation manager list to the newly created role.

5 After the migration is done and verified, remove the obsolete system roles.

Migrating the Auditor Assignments

After the User Application driver migration has been completed, you need to assign each Auditor identity from the list you compiled to be a Role Manager and configure the permissions for the identity. In order to setup the same permission set as in 3.6.1, each identity from the list has to get all of the reporting navigation permissions, as well as the Report On Role and Report On Sod permissions for the role configuration container.

Follow these steps to migrate the Auditor assignments:

1 Login to the User Application as a Security Administrator.

2 Navigate to *Administration>RBPM Provisioning and Security*.

3 Select Administrator Assignments in the left navigation menu.

4 Migrate assigned identities by repeating the following steps for each identity in the list you compiled before the migration:

4a Select *Assign* to open the *New Administrator Assignment* dialog.

4b Select the *Role* domain.

4c Select identity type (user, group, container or role) and lookup the identity.

4d Uncheck the *All Permissions* check box.

4e Click *Assign*.

4f Do not close the *New Administrator Assignments* dialog.

Next, you need to configure the permissions for each user. There was no data security available for audit reporting in 3.6.1. The 3.7 release exposes the ability to secure data by specifying the Report on Role and Report on SoD permissions. In order to setup the same permission set as in 3.6.1, each identity from the list has to get all reporting navigation permissions and the Report On Role and Report On Sod permissions for all roles and SoDs. All report permissions and report items navigation permission are granted out of the box via the Role Manager assignment.

4g In the *Permissions Configuration* section, select *New* to open the Permissions dialog.

4h In the *Add Role Permissions* section, select the Report On Role permission and select all role levels.

4i In the *Add Separation of Duties Permissions* section, select the Report On SoD permission and select *All Separation of Duties Constraints*.

4j Click the *Save* button.

4k Close the *New Administrator Assignment* dialog by clicking the *Close* button.

4l Process child roles list by creating a new non-system role, and assigning all roles from the child role list to this role. Assign all users, groups, containers and roles from the attestation manager list to the newly created role.

5 After the migration is done and verified, remove the obsolete system roles.

Migrating the Team Permissions

After the User Application driver migration has been completed, you need to migrate the team permissions. The process of migrating team permissions involves setting permissions for the newly created (migrated) teams in the User Application. Alternatively, you can create an LDIF generator utility and run it (as described in Section 1.4.4, “Sample Code,” on page 32).

To migrate team permissions:

1. Login to the User Application as a Security Administrator.
2. Navigate to *Administration>RBPM Provisioning and Security*.
3. Select *Team Configuration* in the left navigation menu.
4. Select a team and click *Edit*.
5. In the *Permissions* section, click *New*.
6. Under the *Add Provisioning Request Definition Permissions* section, select the *Permissions* that correspond with the permission settings recorded during pre-migration for the current team.

7 Review the table below to determine the equivalent permission names.

RBPM 3.6.1			RBPM 3.7			
Object Class	**Attribute**	**Permission Description**	**Permission Name**	**nrfAccessAttribute**	**ACL set on**	**ACL trustee**
srvprvTeam	srvprvAllowMgrSetProxy	Allow manager to set proxy for team members	Configure Proxy	nrfAccessProxyConfigure	ACLS should be set on cn=RequestDefs,cn=AppConfig,<your user application driver>. If srvprvRequest attributes are defined, it should be set on srvprvRequest instance dn. If srvprvCategoryKey is defined, the ACLs should be set on all srvprv instances that belong to the category. If you select all provisioning requests, the ACL should be set on cn=RequestDefs,cn=AppConfig, <your user application driver>	srvprvRbpmTeam dn
	srvprvAllowMgrSetAvailability	Allow manager to set availability of team members	Configure Availability	nrfAccessAvailabilitySet		
srvprvTeamRequest	srvprvAllowMgrInitiate	Allow managers to initiate a request for a team member	Initiate PRD	nrfAccessMgrInitiate		
	srvprvAllowMgrSetDelegate	Allow managers to make a team member a delegatee for other team member's requests	Configure Delegate	nrfAccessDelegateConfigure		
	srvprvAllowMgrRetract	Allow managers to retract a request for a team member	Retract PRD	nrfAccessMgrRetractPRD		
	srvprvAllowMgrTaskReassign	Allow managers to reassign a task for team members who are recipients based on task scope	Manage Addressee Task	nrfAccessMgrTaskAddressee		
	srvprvAllowMgrTaskClaim	Allow managers to claim a task for team members who are an addressee				

RBPM 3.6.1			RBPM 3.7			
Object Class	Attribute	Permission Description	Permission Name	nrfAccessAttribute	ACL set on	ACL trustee
	srvprvTaskScopeRecipient	Allow managers to claim a task for team members who are recipients and/or addressees based on the task scope	N/A	N/A	N/A	

1.4.4 Sample Code

This section includes sample Java code that may facilitate the migration process. Topics include:

- "Sample Java Code to Construct a List of Role Members" on page 32
- "Sample Java Code to Get a List of Objects for Which an Identity is Trustee" on page 33
- "Sample Java Code to Generate the ACL LDIF for a Specified Identity" on page 36
- "Constants Used in Sample Code" on page 37

Sample Java Code to Construct a List of Role Members

Here is some sample Java code you might use to construct a list of identities that are assigned to a particular role:

```
List<String> getAllUsraAssigned(LdapContext ldCtx, String role, String rootDn)
throws NamingException
    {
        List<String> identities = new ArrayList<String>();
        SearchControls ctls = new SearchControls();
        ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);
        ctls.setReturningAttributes(new String[]{"nrfMemberOf"});
        String objectClass = "nrfIdentity";
        String attributeName = "nrfMemberOf";
        String searchRoot = rootDn;
        String filter = "(&(objectClass=" + objectClass + ")(nrfMemberOf=" +
role + "))";
        //String driver = "cn=PicassoDriver,cn=TestDrivers,o=novell";
        NamingEnumeration<SearchResult> results = ldCtx.search(searchRoot,
                filter, ctls);
        while (results != null && results.hasMore()) {
            SearchResult nextEntry = results.next();
            Attributes attrs = nextEntry.getAttributes();
            //object dn
            String dn = nextEntry.getNameInNamespace();
            identities.add(dn);
        }
        return identities;
    }
```

Sample Java Code to Get a List of Objects for Which an Identity is Trustee

Here is some sample Java code you might use to get a list of objects for which a particular identity is set as trustee of the ACL for entry browse rights:

```
/**
     * @param ldCtx       - ldap context to use for IDVault connection
     * @param id          - identity id ( user dn)
     * @param objectType: type of the objects to cjheck ACLs on: 1 -role, 2 -
sod, 3 - reports
     * @param driver      - user application drver dn.
     * @return list of object dns that identity dn is trustee
     */
    private List<String> getAllObjectsIdentityIsTrustee(final LdapContext
ldCtx, final String id, final int objectType, final String driver) throws
Exception
    {
        List<String> objects = new ArrayList<String>();

String objectClass = "nrfRole";
        String objectCntrClass = "nrfRoleDefs";
        if (objectType == 2) {
            objectClass = "nrfRole";
            objectCntrClass = "nrfRoleDefs";
        }
        objects = findAllAuthorizationsByObjectClassAndIdentity(ldCtx,
objectClass, id, driver, objectCntrClass, ENTRY_RIGHTS);
        return objects;
    }

 /**
     * Return list of authorizaed objects
     *
     * @param lctx         lCtx       - LDAP context.
     * @param objectClass  - object class
     * @param identity     - identity DN.
     * @param contObjClass - object containet class
     * @return map of ACls keyed by authorized object DN for specified object
class and identity.
     * @throws Exception in case of an error;
     */
    public List<String>
findAllAuthorizationsByObjectClassAndIdentity(LdapContext lctx, String
objectClass,
                                                                    String
identity, final String searchRoot, final String contObjClass, String attr)
            throws Exception
    {
        List<String> authorizedObjects = new ArrayList<String>();
        List<String> containers = new ArrayList<String>();
        SearchControls ctls = new SearchControls();
        ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);
        ctls.setReturningAttributes(new String[]{"ACL"});
        String currSearchRoot = searchRoot;

        NamingEnumeration<SearchResult> results = lctx.search(currSearchRoot,
                "(objectClass=" + objectClass + ")", ctls);
        //Search each group in each team
        while (results != null && results.hasMore()) {
            SearchResult nextEntry = results.next();
            Attributes attrs = nextEntry.getAttributes();
            //object DN
            String dn = nextEntry.getNameInNamespace();
            updateContainerList(dn, containers, true);
            Attribute acl = attrs.get("ACL");
            if (acl != null) {
                String attrValue;
                NamingEnumeration<?> attrValues = acl.getAll();
                while (attrValues.hasMore() &&
                        (attrValue = (String) attrValues.next()) != null) {
                    //trustee for current identity
                    if (attrValue.contains(identity) && (null == attr ||
attrValue.contains(attr))) {
                        authorizedObjects.add(dn);
                    }
                }

            }
        }


        //query for containers

        if (null != contObjClass) {
            results = lctx.search(currSearchRoot,
                    "(objectClass=" + contObjClass + ")", ctls);
            while (results != null && results.hasMore()) {
                SearchResult nextEntry = results.next();
                //object DN
                String dn = nextEntry.getNameInNamespace();
                updateContainerList(dn, containers, false);
            }
        }
        try {
            if (containers.size() > 0) {
                for (String cont : containers) {
                    //get all ACLs for containers and identity
                    List<String> authContainers = readEntryBrowseACLs(lctx,
cont, identity);
                    if (null != authContainers && authContainers.size() > 0) {
                        authorizedObjects.addAll(authContainers);
                    }
                }
            }
        } catch (Throwable th) {
            th.printStackTrace();
        }

        return authorizedObjects;
    }

    private void updateContainerList(final String dn, List<String> containers,
boolean stripFirst) throws InvalidNameException
    {
        LdapName ln = new LdapName(dn);
        int contNambers = ln.getRdns().size() - (stripFirst ? 1 : 0);
        for (int i = contNambers; i > 0; i--) {
            Name cont = ln.getPrefix(i);
            if (!containers.contains(cont.toString())) {
                containers.add(cont.toString());
            }
        }
    }


    /**
     * Read ACL and parse into list of ACL keyed by trustee DN.
     *
     * @param lCtx       - LDAP context.
     * @param objectdn   - authorized object DN.
     * @param identityDn - DN of identity
     * @return list of ACL keyed by trustee DN.
     * @throws IDMAuthorizationException in case of an error.
     */
    public List<String> readEntryBrowseACLs(LdapContext lCtx, String objectdn,
final String identityDn) throws IDMAuthorizationException, NamingException
    {
        assert lCtx != null : "Ldap context is undefined";
        assert objectdn != null : "Authorized object dn is undefined.";
        List<String> authorized = new ArrayList<String>();
        // return entry DN's ACL attribute
        String returnAttrs[] = {ACL_ATTR};
        // get ENTRYDN's ACL attributes

        Attributes attrs = lCtx.getAttributes(objectdn, returnAttrs);
        // get Enumeration of returned attributes (only ACL)
        NamingEnumeration<? extends Attribute> ae = attrs.getAll();
        // parse out ACL attributes
        while (ae.hasMore()) {
            Attribute attr = ae.next();
            NamingEnumeration<?> attrValues = attr.getAll();
            String attrValue;
            while (attrValues.hasMore() &&
                    (attrValue = (String) attrValues.next()) != null) {
                if (null == identityDn || attrValue.contains(identityDn)) {
                    ACL curr = parseACLValue(attrValue, objectdn);
                    if (curr.getPropName() != null &&
curr.getPropName().equals(ENTRY_RIGHTS) &&
                            ((curr.getPriviliges() &
LDAPDSConstants.LDAP_DS_ENTRY_BROWSE) != 0 ||
                                    (curr.getPriviliges() &
LDAPDSConstants.LDAP_DS_ATTR_SUPERVISOR) != 0)) {

                    }
                    authorized.add(objectdn);
                }
            }
        }
        return authorized;
    }


    private ACL parseACLValue(final String aclValue, final String objDn)
    {
        ACL currACL = new ACL();
        String scope, trusteeName, propName, strPriviliges;
        int privileges;
        // ACL value format: "privileges#scope#subjectname#protectedattrname".
        strPriviliges = aclValue.substring(0, aclValue.indexOf('#'));
        privileges = Integer.parseInt(strPriviliges);
        propName = aclValue.substring(
                aclValue.lastIndexOf('#') + 1, aclValue.length());

        //create authorization
        // truncate ACL value to "scope#subjectname"
        String truncACLValue = aclValue.substring(
                aclValue.indexOf('#') + 1, aclValue.lastIndexOf('#'));
        scope = truncACLValue.substring(0, truncACLValue.indexOf('#'));
        trusteeName = truncACLValue.substring(
                truncACLValue.indexOf('#') + 1, truncACLValue.length());
        currACL.setObject(objDn);
        currACL.setTrustee(trusteeName);
        currACL.setScope(scope);
        currACL.setPropName(propName);
        currACL.setPriviliges(privileges);

        return currACL;

    }
```

Sample Java Code to Generate the ACL LDIF for a Specified Identity

Here is some sample Java code you might use to generate the ACL LDIF for a specified identity:

```
/**
     * @param id
     * @param objects
     * @param permissionType
     * @param objectType
     * @return current ACL string buffer
     */
    private StringBuffer generateACLLdif(final LdapContext ldCtx, final String
id, final List<String> objects, String[] accessAttributes) throws
NamingException
    {
        StringBuffer currACL = new StringBuffer();
        System.out.println("Generating ACL for: " + id);
        String scope = SCOPE;

        if (null != objects && null != id) {
            for (String object : objects) {
                if (CONTAINER == getObjectType(object, ldCtx) ||
                        CONTAINER == getObjectType(id, ldCtx)) {
                    //use inhireted
                    scope = SUBTREE_SCOPE;
                }

                if (null != accessAttributes && accessAttributes.length > 0) {
                    currACL.append("dn: ").append(object).append("\n");
                    currACL.append("changetype: modify").append("\n");
                    currACL.append("add: ACL").append("\n");
                    for (String accessAttr : accessAttributes) {
                        currACL.append("ACL:
4#").append(scope).append("#").append(id).append("#").append(accessAttr).appe
nd("\n");
                    }
                    currACL.append("\n");
                }
            }
        }
        return currACL;
    }


 /**
     * Determine object type.5
     *
     * @param dn   object instance key
     * @param lctx Bound LDAP context
     * @return int specifying the object type:  1- USER, 2- GROUP, 3- ROLE, 4
- SOD, 5- CONTAINER, 6-OTHER).
     * @throws NamingException Error reading DN
     */
    private int getObjectType(String dn, LdapContext lctx) throws
NamingException
    {
        Attributes attributes = lctx.getAttributes(dn, ATTR_LIST_OBJ);
        Attribute strClass = attributes.get(ATTR_OBJECT_CLASS);

        if ( strClass != null && strClass.contains(USER_OBJECT_CLASS)) {
            return USER;
       } else if ( strClass != null && strClass.contains(GROUP_OBJECT_CLASS))
{
            return GROUP;
       } else if ( strClass != null && strClass.contains(ROLE_OBJECT_CLASS)) {
            return ROLE;
        } else if ( strClass != null && strClass.contains(SOD_OBJECT_CLASS)) {
            return SOD;
        } else {
            Attribute entryFlags = attributes.get(ATTR_ENTRY_FLAGS);
            if (entryFlags != null) {
                int flags = Integer.parseInt(entryFlags.get().toString());
                if ((flags & LDAPDSConstants.LDAP_DS_CONTAINER_ENTRY) != 0) {
                    return CONTAINER;
                }
            }
        }
        return OTHER;
    }
```

Constants Used in Sample Code

The following listing shows the constants used in the sample code presented above:

```
 public static final String ENTRY_RIGHTS = "[Entry Rights]";
    private static final String SUBTREE_SCOPE = "subtree";
    private static final String ACL_SEPARATOR = "#";
    private static final String SCOPE = "entry";
    private static final String ACL_ATTR = "acl";
    private static final String NRF_ACCESS_PREFIX = "nrfAccess";
    private static final String ALL_ATTRIBUTES = "[All Attributes Rights]";
    private static final String PUBLIC_USER = "[Public]";
    private static final String ATTR_OBJECT_CLASS = "structuralObjectClass";
    private static final String ATTR_ENTRY_FLAGS = "entryFlags";
    private static final String[] ATTR_LIST_OBJ = new 
String[]{ATTR_OBJECT_CLASS, ATTR_ENTRY_FLAGS};
    public static final String ROLE_OBJECT_CLASS = "nrfRole";
    public static final String SOD_OBJECT_CLASS = "nrfSod";
    public static final String USER_OBJECT_CLASS = "inetOrgPerson";
    public static final String GROUP_OBJECT_CLASS = "groupOfNames";
    public static final int USER = 1;
    public static final int GROUP = 2;
    public static final int ROLE = 3;
    public static final int SOD = 4;
    public static final int CONTAINER = 5;
    public static final int OTHER = 6;
    public static final String NRF_CREATE = "nrfAccessCreateRole";
    public static final String NRF_VIEW = "nrfAccessViewRole";
    public static final String NRF_UPDATE = "nrfAccessUpdateRole";
    public static final String NRF_DELETE = "nrfAccessDeleteRole";
    public static final String NRF_ASSIGN = "nrfAccessMgrAssignRole";
    public static final String NRF_REVOKE = "nrfAccessMgrRevokeRole";
    public static final String NRF_ASSIGN_IMPL = "nrfAccessMgrAssignRoleImpl";
    public static final String NRF_REVOKE_IMPL = "nrfAccessMgrRevokeRoleImpl";
    public static final String NRF_REPORT = "nrfAccessReportOnRole";

    public static final String[] ROLE_ACCESS_ATTRS = new String[]{NRF_VIEW, 
NRF_UPDATE,
            NRF_DELETE, NRF_ASSIGN, NRF_REVOKE, NRF_ASSIGN_IMPL, 
NRF_REVOKE_IMPL, NRF_REPORT};

    public static final String NRF_CREATE_SOD = "nrfAccessCreateSoD";
    public static final String NRF_VIEW_SOD = "nrfAccessViewSoD";
    public static final String NRF_UPDATE_SOD = "nrfAccessUpdateSoD";
    public static final String NRF_DELETE_SOD = "nrfAccessDeleteSoD";
    public static final String NRF_REPORT_SOD = "nrfAccessReportSoD";

    public static final String[] SOD_ACCESS_ATTRS = 
            new String[]{NRF_VIEW_SOD, NRF_UPDATE_SOD, NRF_DELETE_SOD, 
NRF_REPORT_SOD};

    public static final String[] REPORT_ACCESS_ATTRS = 
            new String[]{NRF_REPORT, NRF_REPORT_SOD};

    public static final String[] REPORT_SOD_ACCESS_ATTRS = 
            new String[]{NRF_REPORT_SOD};
```