NetIQ® Identity Manager

Administrator's Guide to the Identity Applications

March 2020

Legal Notice

For information about NetIQ legal notices, disclaimers, warranties, export and other use restrictions, U.S. Government restricted rights, patent policy, and FIPS compliance, see https://www.netiq.com/company/legal/.

Copyright (C) 2020 NetIQ Corporation. All rights reserved.

Contents

About this Book and the Library 17
About NetIQ Corporation 19

Part I Overview 21

1 Introduction to the Individual Identity Applications Components 23

Identity Manager Dashboard . . . 23
Identity Applications Administration . . . 24
Understanding Roles . . . 24
Understanding Resources . . . 25
Understanding Separation of Duties . . . 25
Understanding Email-based Approval . . . 25
Understanding Controlled Permission Reconciliation Services . . . 26
Understanding Entities . . . 26
Identity Manager Client Settings . . . 26
Identity Manager Workflows . . . 27
Identity Reporting . . . 27
Identity Applications Security and Password Management . . . 27
User Application . . . 27

2 Types of User Categories in Identity Applications 29

Administrative Users . . . 29
Identity Vault Administrator . . . 29
User Application Administrator . . . 30
Administrator and Manager Categories . . . 30
Understanding Domain Administrators and Managers . . . 31
Designers . . . 33
Business Users . . . 33

3 Understanding the Functionality of the Identity Applications 35

Enabling Self-Service Activities for Users . . . 35
Providing Permissions to Users . . . 36
Understanding Workflow-Based Provisioning . . . 37
Understanding a Client Helpdesk . . . 37
Ensuring Permission Assignments Comply with Your Standards . . . 38
Design and Configuration Tools . . . 38

4 Understanding the Back-end Functions for the Identity Applications 41

User Interfaces . . . 42
Directory Abstraction Layer . . . 43
Workflow Engine . . . 43
SOAP Endpoints . . . 43

Application Server . . . 44
Database . . . 45
User Application Driver . . . 45
Role and Resource Service Driver . . . 46
Multi-Threaded Role and Resource Service Driver . . . 47
Designer for Identity Manager . . . 49
iManager . . . 49
Identity Manager Engine . . . 49
Identity Vault . . . 49

Part II Preparing the Identity Applications for Use 51

5 Understanding the Design Needs 53

Design Constraints . . . 53
High Availability Design . . . 54

6 Configuring Security in the Identity Applications 55

Understanding Security in the Identity Applications Environment . . . 55
Using Secure Sockets for User Application Connections to the Identity Vault . . . 57
Disabling Secure Communications Using the Configuration Update Utility . . . 57
Disabling Secure Communications Using iManager . . . 58
Enabling SSL for User Access . . . 58
Enabling SOAP Security . . . 58
Enabling Authentication . . . 59
Enabling Mutual Authentication . . . 59
Enabling Third-Party Authentication and Single Sign-On . . . 59
Encrypting Sensitive Identity Applications Data . . . 59
Preventing XSS Attacks . . . 60
Modifying Trustee Rights . . . 60
Modifying the Trustee Rights for User Preferences . . . 60
Modifying the Trustee Rights for a Provisioning Request Definition . . . 61
Restricting a User from Viewing Provisioning Request Definitions and Roles in Identity Applications . . . 62
Updating a Password for a Database User on Tomcat . . . 62

7 Assigning the Identity Applications Administrators 63

Understanding the Administrators of the Identity Applications . . . 63
Changing the Default Administrator Assignments after Installation . . . 64
Granting or Removing Assignments in the User Application . . . 65
Changing the Assignments in Configupdate Utility . . . 65
Changing the Default Administrator Assignments without an Administrator Account . . . 66

8 Setting Up Logging in the Identity Applications 67

How Logging Services Help . . . 67
What Can Be Logged . . . 75
How Logging Works . . . 76
Terminology . . . 76

Components for Logging . . . 77
How Logging Works . . . 77
Types of Log Files . . . 78
Difference Among Catalina, Application, and Localhost Log Files . . . 79
Additional Log Files . . . 80
Understanding the Log Format . . . 81
Message Fields . . . 81
Message Severity . . . 82
Configuring Logging . . . 83
Understanding Logging Configuration . . . 84
Understanding the Log Level Settings . . . 84
Specifying the Severity Level for Commons Logging API Loggers . . . 85
Configuring Logging Settings in Identity Manager Dashboard . . . 85
Editing the log4j Files . . . 86
Managing Log File Size . . . 87
Configuring Logging in a Cluster . . . 87
Tomcat Logging . . . 88
User Application Logging . . . 88
Logging to a Sentinel Server . . . 88
Configuring the Platform Agent . . . 89
Enabling Sentinel Logging . . . 89
Using Log Files for Troubleshooting . . . 90
Log Events . . . 90

9 Tuning the Performance of the Applications 97

Increasing the Heap Size . . . 99
Increasing the Stack Size for Recursive Workflows . . . 99
Ensuring Concurrent Access from Multiple Clients . . . 100
Decreasing the Session Time-out . . . 100
Increasing the Number of Maximum Open Files . . . 101
Increasing the Number of User Processes . . . 102
Adjusting the Threadpool Size . . . 102
Increasing the Database Connection Pool . . . 103
View Request Status Search Limit . . . 103
Decreasing the LDAP Socket Cleanup Interval . . . 104
Optimizing LDAP Connection with Identity Vault . . . 104
Indexing Attributes in the Identity Vault . . . 105
Enabling Compound Index on Identity Vault Attributes . . . 106
Comparison with Other Indexes . . . 108
Sample Error Message . . . 108
Managing the eDirectory Database Cache Objects Retrieved from the Identity Vault Server . . . 108

10 Customizing the Identity Applications for Your Enterprise 111

Linking the Dashboard to External Applications . . . 111
Managing Featured Items . . . 112
Customizing the Look of the User Interfaces . . . 112
Applying Your Organization’s Brand to the Dashboard . . . 112
Applying a Cascading Style Sheet to the Dashboard . . . 113
Localizing the Text in the Interfaces . . . 113
Localizing the Labels in the Dashboard . . . 113
Hiding the Applications Tab . . . 115

Modifying the Text of the Application Tab . . . 115
Localizing the Text Stored in the JAR Files . . . 116
Adding a Language to the Identity Applications . . . 119
Adding the New Language to the Identity Applications . . . 120
Preparing Files for Translation . . . 121
Changing the Default Language . . . 123
Add the Translated Files to the Proper Locations . . . 124
Updating an Email Notification Template . . . 125
Verifying the New Translations . . . 125
Configuring User Names . . . 125
Configuring the Format of Displayed User Names . . . 126
Enabling Localized User Names in Typeahead Fields . . . 126
Configuring Email Notification Templates for the Dashboard . . . 127
Configuring Forgot Password? Functionality . . . 128
Ensuring that Characters Display Properly in Role Report PDF Files . . . 128
Editing the Configuration XML Data in iManager . . . 129
Ensuring that Dates Display Correctly in Norwegian . . . 129
Configuring Client Settings Mode . . . 130
Changing Identity Applications Client Settings . . . 131
Changing General Client Settings . . . 131
Customizing the Views . . . 131
Managing User Access . . . 135
Changing the Client Branding Attributes . . . 136
Configuring a Client Helpdesk . . . 136
Managing Dashboard Widgets . . . 139
Deleting the Client Settings from Identity Applications . . . 139

11 Setting Up the Dashboard for Identity Applications 141

Checklist for Setting Up the Dashboard for Identity Applications . . . 141

12 Configuring a Multi-Threaded Role and Resource Service Driver 143

How the Driver Works . . . 143
Prerequisites . . . 144
Defining a Unique Data Set . . . 144
Modifying the Default Mapping Table Object . . . 145
Configuring the Driver . . . 145
Deploying the Driver . . . 146
Limitations . . . 146
Troubleshooting . . . 146

13 Configuring Identity Applications Clustering and Permission Clustering 147

Configuring Identity Applications Clustering to Use TCP or UDP . . . 147
Configuring Permission Clustering to Use TCP or UDP . . . 148

Part III Identity Applications Administration 151

14 Creating and Managing Roles 153

Listing Roles . . . 153

Creating a New Role 154
Editing Roles 154
Changing Approval and Revocation Process 156
Mapping Resources to Roles 156
Assigning Roles to Users 157
Mapping Roles to Roles 158
Managing the Role and Resource Service Driver 159
Configuring the Role and Resource Service Driver Settings 159
Indexing for the Role and Resource Service Driver 160

15 Creating and Managing Resources 163

Listing Resources 163
Creating a New Resource 164
Editing Resources 165
Setting Expiration Period for the Resource 166
Changing the Approval or Revocation Process 166
Assigning Resource to Users 167
Updating the Resource Request Form 167
Enabling Drivers for Resource Mappings 168
Creating a List to Improve Resource Request Forms 169
Resource Assignments 170
Resource Request Process Flow 171

16 Creating and Managing Delegations 173

17 Separation of Duties Constraints 175

18 Using Controlled Permission Reconciliation Services 177

How CPRS Helps 177
Prerequisites 178
Considerations for Supported Drivers 178
MDAD Driver 178
Loopback Driver 179
REST Driver 179
Delimited Text Driver 179
Understanding the Components of CPRS 179
Managing Permission Reconciliation Settings 181
Editing Permission Reconciliation Settings 182
Permission Reconciliation 183
Migrating to CPRS 185
Prerequisites 185
Managing Existing Permissions for AD and LDAP Drivers 186
Managing Permissions for a MDAD Driver 186
Post Migration Activities 187

19 Configuring Identity Applications Default Settings 189

Configuring Roles and Resources Settings 189
Configuring Default Roles Settings 189
Configuring Default Resource Settings 190

Configuring Entitlement Query Settings 190
Configuring Separation of Duties Settings 191
Configuring Delegation and Proxy Settings 191
Configuring Permission Reconciliation Settings 191
Configuring Logging Settings 192
Configuring Auditing Service Settings 192
Configuring the Identity Manager Packages and their Log Levels 193
Configuring Caching and Cluster Settings 193
Flushing Caches 193
Configuring Cache Settings 194
Managing Cluster Cache Settings 199
Assigning Administrators in Identity Applications 199
Listing the Administrator Assignments 200
Creating a New Administrator Assignment 200
Assigning Permissions to a Delegated Administrator 201
Deleting an Administrator Assignment 204
Configuring Workflow Engines and Cluster Settings 204
Configure the Workflow Engine Settings 205
Configure Workflow Cluster Settings 206
Viewing User Application Driver Status 207
Configuring the Default Provisioning Display Settings 207
Managing General Display Settings 207
Managing the Appearance of Tasks Page 208
Managing the Appearance of Request History Page 209

20 Configuring Email-Based Approval 211

21 Configuring and Managing Objects for Entities 213

Listing the Objects 213
Creating an Object 213
Editing an Object 214
Deleting an Object 214
Exporting to CSV 214

Part IV Configuring and Managing Provisioning Workflows 215

22 Configuring the User Application Driver to Start Workflows 217

About the User Application Driver 217
Setting Up Workflows to Start Automatically 218
About Policies 218
Using the Policy Builder 218

23 Managing Provisioning Request Definitions 223

About the Provisioning Request Configuration Plug-in 223
Working with the Installed Templates 224
Configuring a Provisioning Request Definition 227
Selecting the Driver 227
Deleting a Provisioning Request 227
Filtering the List of Requests 228

Changing the Status of an Existing Provisioning Request 229
Defining Rights on an Existing Provisioning Request 229

24 Managing Provisioning Workflows 231

About the Workflow Administration Plug-in. 231
Managing Workflows 231
Connecting to a Workflow Server. 232
Restricting Access to Workflows. 233
Finding Workflows that Match Search Criteria 233
Controlling the Active Workflows Display 234
Terminating a Workflow Instance. 236
Viewing Details about a Workflow Instance 236
Reassigning a Workflow Instance 237
Managing Workflow Processes in a Cluster. 237
Configuring the Email Server 239
Working with Email Templates. 240
Default Content and Format 241
Editing Email Templates 252
Modifying Default Values for the Template 253
Adding Localized Email Templates 253
Allowing a Named Password to be Retrieved over LDAP. 254

Part V Web Service Reference 257

25 Provisioning Web Service 259

About the Provisioning Web Service 259
Provisioning Web Service Overview. 259
Removing Administrator Credential Restrictions 260
Provisioning Web Service Method Categories 260
Developing Clients for the Provisioning Web Service 261
Web Access to the Provisioning Web Service 261
A Java Client for the Provisioning Web Service 263
Developing a Mono Client. 268
Sample Ant File 269
Sample Log4J File. 271
Provisioning Web Service API. 271
Processes 271
Provisioning 282
Work Entries. 295
Comments 313
Configuration 320
Miscellaneous 324
Cluster. 327

26 Metrics Web Service 331

About the Metrics Web Service 331
Web Service Semantics 332
Accessing the Test Page. 332
Web Service Methods Grouped by Security Permissions. 332
Specifying Filters 335

Generating the Stub Classes . . . 337
Obtaining the Remote Interface . . . 338
Metrics Configuration Settings . . . 339
Metrics Web Service API . . . 341
Team Manager Methods . . . 341
Provisioning Application Administrator Methods . . . 343
Utility Methods . . . 345
Metrics Web Service Examples . . . 346
General Examples . . . 346
Other Examples . . . 348

27 Notification Web Service 351

About the Notification Web Service . . . 351
Accessing the Test Page . . . 351
Accessing the WSDL . . . 351
Generating the Stub Classes . . . 352
Notification Web Service API . . . 352
iRemoteNotification . . . 352
BuiltInTokens . . . 352
Entry . . . 354
EntryArray . . . 355
NotificationMap . . . 356
NotificationService . . . 357
StringArray . . . 357
VersionVO . . . 358
Notification Example . . . 358

28 Directory Abstraction Layer (VDX) Web Service 361

About the Directory Abstraction Layer (VDX) Web Service . . . 361
Accessing the Test Page . . . 361
Accessing the WSDL . . . 361
Generating the Stub Classes . . . 362
Removing Administrator Credential Restrictions . . . 362
VDX Web Service API . . . 363
IRemoteVdx . . . 363
Attribute . . . 365
AttributeArray . . . 367
AttributeType . . . 368
BooleanArray . . . 368
ByteArrayArray . . . 369
DateArray . . . 369
EntryAttributeMap . . . 370
Entry . . . 370
EntryArray . . . 371
IntegerArray . . . 372
StringArray . . . 373
StringEntry . . . 373
StringEntryArray . . . 374
StringMap . . . 375
VdxService . . . 375
VersionVO . . . 375
VDX Example . . . 376

29 Role Web Service **387**

About the Role Web Service 387
Accessing the Test Page 387
Accessing the WSDL 390
Generating the Stub Classes 390
Removing Administrator Credential Restrictions 390
Role API 391
IRemoteRole 391
Approver 407
ApproverArray 408
Category 408
CategoryArray 409
CategoryKey 410
CategoryKeyArray 410
Configuration 411
Container 414
DNString 415
DNStringArray 416
Entitlement 417
EntitlementArray 418
Group 418
IdentityType 420
IdentityTypeDnMap 422
IdentityTypeDnMapArray 423
LocalizedValue 423
LongArray 424
NrfServiceException 424
RequestCategoryType 425
RequestStatus 427
ResourceAssociation 429
Role 431
RoleAssignment 436
RoleAssignmentArray 438
RoleAssignmentActionType 439
RoleAssignmentRequest 440
RoleAssignmentRequestStatus 443
RoleAssignmentType 447
RoleAssignmentTypeInfo 448
RoleInfo 451
RoleInfoArray 452
RoleLevel 453
RoleLevelArray 454
RoleRequest 455
RoleServiceDelegate 458
RoleServiceSkeletonImpl 463
Sod 467
SodArray 470
SodApprovalType 471
SodJustification 472
SodJustificationArray 473
User 474
VersionVO 478
Role Web Service Examples 479
Retrieving Roles for a Group 479

Retrieving Role Assignment Request Status 481
Retrieving Type Information for a Role Assignment 482
Retrieving Role Categories 483
Retrieving Role Levels 484
Verifying Whether a User Is In a Role 484

30 Resource Web Service 487

About the Resource Web Service 487
Accessing the Test Page 487
Accessing the WSDL 488
Generating the Stub Classes 489
Removing Administrator Credential Restrictions 489
Resource Web Service Interface 490
IRemoteResource 490
CodeMapRefreshStatus 504
CodeMapValueStatus 505
EntitlementRefreshInfo 506
ProvisioningCodeMap 507
Resource 510
ResourceAssignment 514
ResourceRequestParam 516
ResourceAssignmentRequestStatus 517
Resource Web Service Examples 520
Code Map Synchronization Code Samples 520

31 Forgot Password Web Service 523

About the Forgot Password Web Service 523
Accessing the Service 523
Accessing the WSDL 523
Generating the Stub Classes 524
Password Management Web Service Interface 524
processForgotConf 524
processUser 525
processChaRes 525
processChgPwd 526
ForgotPasswordWSBean 527

Part VI Configuring Single Sign-on Access in Identity Manager 529

32 Preparing for Single Sign-on Access 531

33 Using Self-Service Password Management in Identity Manager 533

Understanding the Default Self-Service Process 533
Understanding the Legacy Password Management Provider 534
Understanding Authentication with One SSO Provider 534
How OSP Works with Identity Manager 535
OSP Concepts 536
Understanding How OSP Works with Identity Manager 538
Guidelines for Enabling OSP Logging 540

34 Using One SSO Provider for Single Sign-on Access in Identity Manager 543

Preparing eDirectory for Single Sign-on Access 543
Modifying the Basic Settings for Single Sign-on Access 543
Configuring Self Service Password Reset to Trust OSP 544

35 Using NetIQ Access Manager for Single Sign-On 545

Understanding Third-Party Authentication and Single Sign-On 545
Using SAML Authentication for Single Sign-on 546
Establishing Trust between Identity Manager and Access Manager 547
Updating the Login Pages for Access Manager 548
Reverse Proxy Based Single Sign-On 549
Creating and Configuring the Proxy Service 552
Creating Protected Resources 554
Creating and Assigning a Form Fill Policy to a Protected Resource 556
Configuring a Rewriter Profile 558
Configuring Identity Providers 559
Configuring Additional Redirect URLs in OSP Configuration File 560
Testing the Single Sign-On 561

36 Using Kerberos for Single Sign-On 563

Configuring the Kerberos User Account in Active Directory 563
Configuring the Identity Applications Server 564
Configure the End-User Browsers to Use Integrated Windows Authentication 566
Logging In Using the Name Password Form 567

37 Integrating Single Sign-on Access with Identity Governance 569

Ensuring Rapid Response to Authentication Requests 569
Configuring Identity Governance for Integration 570
Adding a Link for Identity Manager Home in the Identity Governance Menu 570
Using the Same Authentication Server as Identity Manager 570
Registering Identity Applications Server 572
Configuring Identity Manager for Integration 573

38 Verifying Single Sign-on Access for the Identity Applications 579

39 Using SSL for Secure Communication 581

Checklist for Ensuring SSL Connections 581
Creating a Keystore and Certificate Signing Request 582
Enabling SSL with a External CA Signed Certificate 583
Enabling SSL with a Self-signed Certificate 585
Exporting the Certificate Authority 585
Generating the Self-signed Certificate 586
Enabling SSL Between Sentinel and Identity Manager Components 587
Enabling SSL between Sentinel and Identity Manager Engine/Remote Loader 588
Enabling SSL between Sentinel and User Application 589
Updating the SSL Settings for the Application Server 590
Updating the SSL Settings in the Configuration Utility 592

Updating the SSL Settings for Self Service Password Reset 593

40 REST Services 595

Use Cases for Identity Applications REST API 595
Before You Begin 595
Use Cases 596
Request a Role for Yourself 601

41 Troubleshooting 603

Using Log Files for Troubleshooting. 603
Customizing Logging Settings 603
Virtual Data Access Logging 604
When a Code Map Refresh Is Triggered 607
When Multiple Users Try to Authenticate From Different Interfaces 608
When an E-Mail Approval Notification is Not Delivered 609
When a Role Is Requested 609
When a Role Is Listed in Role Catalog 612
Schema Fails to Update When Updated Using a User Account That Was Not Used to Create the Schema 614
Checking the Status of Database Schema Validation 616
Determining if Liquibase Changeset Has Executed 616
When Assigning a Resource to a User That Does Not Exist 618
When Checking the Workflow Engine Heartbeat 618
catalina.out File Does Not Rotate the Log on Linux 619
Troubleshooting E-Mail Based Approval Issues 620
Empty E-Mail Based Approval Token in the Provisioning Request Mail 620
User Application is Not Acting on E-Mails 620
Approve or Deny Link in E-Mail is Not Working 620
Approve/Deny links Missing from E-Mail after configuring E-Mail Based Approval 620
Verifying if E-Mail Based Approval Starts Properly 620
When is Server Restart Needed 621
E-Mail Based Approval Token is Empty in the Provisioning Request E-Mail 621
Troubleshooting Self Service Password Reset Issues 621
No Redirection to Challenge-Response Page When SSPR is Installed in a Distributed Environment That Supports http and https Communication 621
Unable to Unlock Account through SSPR. 621
SSPR Reports Error 5027 When Attempting to Access Configuration Manager through Internet Explorer 621
SSPR Reports Out of Order Page Request Error 622
Pressing Enter Button in SSPR's People Search Displays Locale Screen on Internet Explorer 622
Troubleshooting Authentication Issues. 622
OSP Login Request Example by Using REST Endpoints 622
Managing the Size of oidPInstancedata Attribute. 624
OSP Fails to Update the oidpInstanceData Attribute 625
Troubleshooting General Issues. 625
Mismatch of Certificates Used by Identity Manager Engine and User Application Causes Code
(-9205) Error in vnd.nds.stream 626
User Application Driver Fails to Communicate with the User Application Server on a Secured Connection 626
Entitlement Configuration Error During Codemap Refresh 627
Error After Logging Out of the Dashboard on Linux 627

Bulk Import of Roles and Resources May Not Update the Permission Index 627
Absence of Notification Templates Causes Workflow Error 628
Error Occurs When You Add a New Application With a Logo 628
User Application Driver Fails to Process Delete Events 629
Card View Showing Only Default Attributes of the Recipient on Tasks Page 629
Cannot Collect Identity Applications Audit Events Through CEF Auditing When SSL Is Enabled 629
Unable to Search for Users While Requesting For Permissions on Behalf of Others 630
Troubleshooting Multi-Threaded Role and Resource Service driver Issues 630

Part VII Appendix 633

A Configuring the Identity Manager Approvals App 635

Product Requirements 635
Setting Up the Approvals App 636
Understanding Approvals App Settings 637
Customizing and Using the Default Approvals App Provisioning Request Definition 639
Creating and Deploying a Custom Configuration Link 644
Creating and Deploying a Custom Configuration QR Code 644
Optimizing Designer Forms for the Approvals App 645
Understanding Language Support in the Approvals App 645

B Schema Extensions for the User Application 647

Attribute Schema Extensions 647
Objectclass Schema Extensions 650
Resource Definition Object (nrfResource) 652
Resource Request Object (nrfResourceRequest) 652
Resource Request Status Codes (nrfStatus) 653
Role Definition Object (nrfRole) 654
Role Status Codes (nrfStatus) 655
Request Object (nrfRequest) 655
Request Status Codes (nrfStatus) 656
Role-Resource Configuration (nrfConfiguration) 657
Resource Binding to Users (nrfIdentity) 658
Resource Containers 658

C JavaScript Search API 659

Launching a Basic Search using the SearchListPortlet 659
Passing Request Parameters 659
Using a JSON-formatted String to Represent a Query 660
Creating a New Query using the JavaScript API 662
JavaScript API 664
Performing an Advanced Search Using a JSON-formatted Query 666
Retrieving all Saved Queries for the Current User 666
Running an Existing Saved Query 666
Performing a Search on All Searchable Attributes 667

D Trouble Shooting **669**

Permgen Space Error 669
Email Notification Templates 669
Org Chart and Guest Access 669
Provisioning Notification 670
javax.naming.SizeLimitExceededException 670
Linux Open Files Error 671

About this Book and the Library

The *Administrator's Guide* describes how to administer the NetIQ Identity Manager using Identity Applications.

Intended Audience

This book provides information for identity architects and identity administrators responsible for installing the components necessary for building an identity management solution for their organization.

Other Information in the Library

For more information about the library for Identity Manager, see the Identity Manager documentation website.

About NetIQ Corporation

We are a global, enterprise software company, with a focus on the three persistent challenges in your environment: Change, complexity and risk—and how we can help you control them.

Our Viewpoint

Adapting to change and managing complexity and risk are nothing new

In fact, of all the challenges you face, these are perhaps the most prominent variables that deny you the control you need to securely measure, monitor, and manage your physical, virtual, and cloud computing environments.

Enabling critical business services, better and faster

We believe that providing as much control as possible to IT organizations is the only way to enable timelier and cost effective delivery of services. Persistent pressures like change and complexity will only continue to increase as organizations continue to change and the technologies needed to manage them become inherently more complex.

Our Philosophy

Selling intelligent solutions, not just software

In order to provide reliable control, we first make sure we understand the real-world scenarios in which IT organizations like yours operate—day in and day out. That's the only way we can develop practical, intelligent IT solutions that successfully yield proven, measurable results. And that's so much more rewarding than simply selling software.

Driving your success is our passion

We place your success at the heart of how we do business. From product inception to deployment, we understand that you need IT solutions that work well and integrate seamlessly with your existing investments; you need ongoing support and training post-deployment; and you need someone that is truly easy to work with—for a change. Ultimately, when you succeed, we all succeed.

Our Solutions

- Identity & Access Governance
- Access Management
- Security Management
- Systems & Application Management
- Workload Management
- Service Management

Contacting Sales Support

For questions about products, pricing, and capabilities, contact your local partner. If you cannot contact your partner, contact our Sales Support team.

Worldwide:	www.netiq.com/about_netiq/officelocations.asp
United States and Canada:	1-888-323-6768
Email:	info@netiq.com
Web Site:	www.netiq.com

Contacting Technical Support

For specific product issues, contact our Technical Support team.

Worldwide:	www.netiq.com/support/contactinfo.asp
North and South America:	1-713-418-5555
Europe, Middle East, and Africa:	+353 (0) 91-782 677
Email:	support@netiq.com
Web Site:	www.netiq.com/support

Contacting Documentation Support

Our goal is to provide documentation that meets your needs. The documentation for this product is available on the NetIQ Web site in HTML and PDF formats on a page that does not require you to log on. If you have suggestions for documentation improvements, click **comment on this topic** at the bottom of any page in the HTML version of the documentation posted at www.netiq.com/documentation. You can also email Documentation-Feedback@netiq.com. We value your input and look forward to hearing from you.

Contacting the Online User Community

NetIQ Communities, the NetIQ online community, is a collaborative network connecting you to your peers and NetIQ experts. By providing more immediate information, useful links to helpful resources, and access to NetIQ experts, NetIQ Communities helps ensure you are mastering the knowledge you need to realize the full potential of IT investments upon which you rely. For more information, visit community.netiq.com.

Overview

Identity applications help you manage different functions of your organization using Identity Manager.

- Chapter 1, “Introduction to the Individual Identity Applications Components,” on page 23
- Chapter 2, “Types of User Categories in Identity Applications,” on page 29
- Chapter 3, “Understanding the Functionality of the Identity Applications,” on page 35
- Chapter 4, “Understanding the Back-end Functions for the Identity Applications,” on page 41

1 Introduction to the Individual Identity Applications Components

Identity applications enable your organization to manage the user accounts and permissions associated with the wide variety of roles and resources available to users. You can configure the identity applications to provide self-service support for your users, such as requesting roles or changing their passwords.

The following components comprise identity applications:

- "Identity Manager Dashboard" on page 23
- "Identity Applications Administration" on page 24
- "Understanding Entities" on page 26
- "Identity Manager Client Settings" on page 26
- "Identity Manager Workflows" on page 27
- "Identity Reporting" on page 27
- "Identity Applications Security and Password Management" on page 27
- "User Application" on page 27

Identity Manager Dashboard

Identity Manager Dashboard serves as the primary entry portal to the identity applications. Dashboard can have one or many widgets that helps you with the quick information on particular activity. From your Dashboard, you can perform the following activities:

- Manage your profile settings and password.
- View your organization chart details.
- Review and complete your tasks, such as approving user requests for access.
- Request permissions for roles, resources, or processes.
- Review the status and history of the requests for permissions.
- Find other users in your organization.
- Personalize your dashboard, you can add widgets and reposition them based on your interests.
- Set any user as your proxy from the system.
- Delegate your tasks to other users from the system.

You can perform the following tasks with the appropriate **Permissions**:

- Create and modify user profiles.
- View the organization chart details of other users.

- Create and modify teams that represent set of users and groups that can perform provisioning requests and approval tasks associated with the teams.
- Request permissions or revoke permissions on behalf of other users in the organization.

Identity Applications Administration

You can manage the following tasks with an appropriate Administrator role:

- Create and manage roles, resources and their assignments.
- Set the Separation of Duties (SoD) constraints to avoid conflicts between two different roles in the system.
- Configure the ability for users to approve permission requests through email.
- Configure the default settings of your identity applications components such as roles, resources, and delegation.

For more information on each identity applications administration options, see:

- "Understanding Roles" on page 24
- "Understanding Resources" on page 25
- "Understanding Separation of Duties" on page 25
- "Understanding Email-based Approval" on page 25
- "Understanding Controlled Permission Reconciliation Services" on page 26

Understanding Roles

A role represents a set of permissions that allows you to perform defined activities using identity applications. A role can be mapped to one or more roles, resources, and entitlements from different connected systems. You can assign any role to any user in your organization.

Identity Manager Dashboard allows you to create and manage role in your organization.

Administration > Roles

You can map role assignments to resources within a company, such as user accounts, computers, and databases. For more information, see Chapter 15, "Creating and Managing Resources," on page 163.

You can modify the default settings for the roles and their operations in the system that can help you to control creating and managing roles.

Administration > Configuration > Roles and Resources

For more information, see "Configuring Default Roles Settings" on page 189.

Understanding Resources

A resource is any digital entity such as a user account, computer, or database that a business user needs to be able to access.

Each entitlement is mapped to a resource. A resource definition can have no more than one entitlement bound to it. A resource definition can be bound to the same entitlement more than once, with different entitlement parameters for each resource.

Identity Manager Dashboard allows you to create and manage resources in your organization.

Administration > Resources

For more information, see Chapter 14, “Creating and Managing Roles,” on page 153.

You can view the default settings for the resources and their operations in the system that controls creating and managing resources.

Administration > Configuration > Roles and Resources

For more information, see “Configuring Default Resource Settings” on page 190.

Understanding Separation of Duties

Separation of duties (SoD) policies help you manage potential conflicts between role assignments. For example, your organization might have two or more roles that could create security problems when assigned to the same individual. When a user requests one of these roles while already having a conflicting role or requests two or more conflicting roles, the identity applications respond according to the SoD policies. For more information, see Chapter 17, “Separation of Duties Constraints,” on page 175.

Understanding Email-based Approval

Identity applications allow to send an email notifying users that they need to review a permission request. The notification can include action links that correspond to Approve and Reject so users can respond to the request. Email-based approvals also supports digital signatures to ensure authentication of the message content.

You enable email-based approvals and configure your Provisioning Request Definitions to support the feature.

Administration > Email Based Approval

For more information, see the following sources:

- click ⓘ in Identity Manager Dashboard.
- “Email Based Approval” in the *NetIQ Identity Manager - Administrator’s Guide to Designing the Identity Applications*

Understanding Controlled Permission Reconciliation Services

Identity applications enable a resource administrator to publish all connected system permission assignments to Identity Manager Resource Catalog through Controlled Permission Collection and Reconciliation Service (CPRS). CPRS helps to keep Resource Catalog up-to-date with connected system permissions at any point of time. For ease of use, CPRS is integrated with the identity applications user interface.

Administration > Permission Reconciliation

For more information, see Chapter 18, “Using Controlled Permission Reconciliation Services,” on page 177.

Understanding Entities

You can customize the User Application by adding objects and their attributes based on the content in the Identity Vault. You can do this by adding new entities and attributes to the directory abstraction layer and deploying them to the User Application driver. For more information, see About Entities and Attributes in *NetIQ Identity Manager - Administrator’s Guide to Designing the Identity Applications*.

Identity Manager Client Settings

Identity Manager allows you to manage one or more clients and helps you to define their attributes. If you have an access to the **Your ID > Settings** page, you can perform the following activities for your clients:

- You can add and manage clients and define the client name and their LDAP properties.
- Set user and configuration accesses.
- Customize the general and user view for your clients and provision your client users with other attributes. You can also enable edit option for more attributes to your client users.
- Define the brand color for your clients which will be displayed for respective client users.
- Set **Helpdesk** to help your client users troubleshoot any issues while performing their tasks in Identity Manager. For more information, see “Understanding a Client Helpdesk” on page 37
- Provision your client users with **Dashboard Widgets** and allow them to personalize their dashboard with provisioned widgets.

For more information about the modes of saving these changes, see “Configuring Client Settings Mode” on page 130.

For more information about changing the client settings, see “Changing Identity Applications Client Settings” on page 131.

Identity Manager Workflows

You can also set up workflows to improve the efficiency in managing and assigning roles and resources. You can perform following tasks with appropriate permissions:

- Create workflows to reduce the administrative burden of entering, updating, and deleting user information across all systems in the enterprise. These workflows provide a Web-based interface for users to manipulate distributed identity data that triggers workflows as necessary. For more information, see Part IV, “Configuring and Managing Provisioning Workflows,” on page 215.
- Support complex workflows and manage manual and automated provisioning of identities, services, resources, and assets.

  You can establish a manual provisioning process by creating workflows that route provisioning requests to one or more authorities. For automated provisioning, you can configure the User Application to start workflows automatically in response to events occurring in the Identity Vault. The Dashboard can trigger a workflow when users request permission. For more information, see Part IV, “Configuring and Managing Provisioning Workflows,” on page 215.

Identity Reporting

As a complement to the identity applications, Identity Manager includes Identity Reporting. If you install Identity Reporting with the identity applications give you a complete view of your users’ entitlements, providing the knowledge you need to see the past and present state of authorizations and permissions granted to identities in your organization.

For more information about reporting, see the *Administrator Guide to NetIQ Identity Reporting*.

Identity Applications Security and Password Management

- Configure password management settings so users can reset their own passwords. For more information, see Password Management Configuration.
- Ensure that your organization has a method for verifying that personnel are fully aware of organizational policies and are taking steps to comply with these policies.
- Ensure that access to corporate resources complies with organizational policies and that provisioning occurs within the context of the corporate security policy. You can grant users access to identity data within the guidelines of corporate security policies. For more information, see Chapter 6, “Configuring Security in the Identity Applications,” on page 55.

User Application

Originally, the User Application was part of the Roles Based Provisioning Module (RBPM). Some of the RBPM functions have been moved to the dashboard interface. The User Application continues to provide the following functions that does not yet exist in the other two components:

- Ensure that your organization has a method for verifying that personnel are fully aware of organizational policies and are taking steps to comply with these policies.

- Ensure that access to corporate resources complies with organizational policies and that provisioning occurs within the context of the corporate security policy. You can grant users access to identity data within the guidelines of corporate security policies. For more information, see Chapter 6, “Configuring Security in the Identity Applications,” on page 55.
- Create workflows to reduce the administrative burden of entering, updating, and deleting user information across all systems in the enterprise. These workflows provide a Web-based interface for users to manipulate distributed identity data that triggers workflows as necessary. For more information, For more information, see Part IV, “Configuring and Managing Provisioning Workflows,” on page 215.
- Support complex workflows and manage manual and automated provisioning of identities, services, resources, and assets.

  You can establish a manual provisioning process by creating workflows that route provisioning requests to one or more authorities. For automated provisioning, you can configure the User Application to start workflows automatically in response to events occurring in the Identity Vault. The Dashboard can trigger a workflow when users request permission. For more information, see Part IV, “Configuring and Managing Provisioning Workflows,” on page 215.

2 Types of User Categories in Identity Applications

Users of the identity applications generally belong to any of the following categories:

- "Administrative Users" on page 29
- "Administrator and Manager Categories" on page 30
- "Designers" on page 33
- "Business Users" on page 33

Administrative Users

The identity applications have several administrative users. During installation, you establish the following administrators:

Identity Vault Administrator

A user who has rights to configure the Identity Vault. This is a logical role that can be shared with other administrative user types.

The Identity Vault Administrator needs the following rights:

- Supervisor rights to the User Application driver and all the objects it contains. You can accomplish this by setting the rights at the driver container level and making them inheritable.
- Supervisor Entry rights to any of the users that are defined through the directory abstraction layer user entity definition. This should include Write attribute rights to objectClass and any of the attributes associated with the `DirXML-EntitlementRecipient`, `srvprvEntityAux` and `srvprvUserAux` auxiliary classes.
- Supervisor rights to the container object `cn=DefaultNotificationCollection`, `cn=Security`. This object persists email server settings used for automated provisioning emails. It can contain SecretStore credentials for authenticating to the email server itself.
- Supervisor rights to the container object `cn=Authorized Login Methods`, `cn=Security`. During the Identity Applications installation the SAML Assertion object is created in this container.
- Ensure that you have supervisor rights to the `cn=Security` container before you install user application. During the Identity Applications installation, the container `cn=RBPMTrustedRootContainer` is created under the `cn=Security` container.

  Alternatively, manually create the `cn=RBPMTrustedRootContainer,cn=Security` container (create an object called `Trusted Root Container` with object class `NDSPKI:Trusted Root` inside the `Security` container), and then assign supervisor rights to the container.

User Application Administrator

A user who has the rights to perform administrative tasks for the identity applications. This user has the following attributes:

- Can manage Identity Manager, identity applications administration, and client settings.
- Can use iManager to administer workflow tasks, such as enabling, disabling, or terminating in-process workflows.
- Does not have any special privileges within the identity applications.
- Does not need any special directory rights because it controls application-level access to the identity applications from the Dashboard. Although a User Application Administrator has the ability to customize the look and feel of the applications, the identity applications use the LDAP administrator credentials to modify the selections in the Identity Vault.
- Can manage the password for this account.

  A feature of password self-service is password synchronization status. To enable the User Application Administrator to view the password synchronization status for other users (for troubleshooting or other reasons), you should create a PasswordManagement group and assign one or more users to this group. The members of this group are allowed to view the password synchronization status of other users. If you choose to create this group, it must:

  - Be named `PasswordManagement`.
  - Be given the privileges to the Identity Vault. The group must have rights to read the user's eDirectory object attribute for users whose password synchronization status they need to view.

IMPORTANT: NetIQ Self Service Password Reset (SSPR) is the default password management program for Identity Manager. For more information, see "Managing Your Password" in the *NetIQ Identity Manager - User's Guide to the Identity Applications*.

Administrator and Manager Categories

Following are the general categories of administrators and managers used in identity applications:

Domain Administrator

An administrator who has the full range of capabilities within a particular domain, and is able to perform all operations on all objects within the domain for all users.

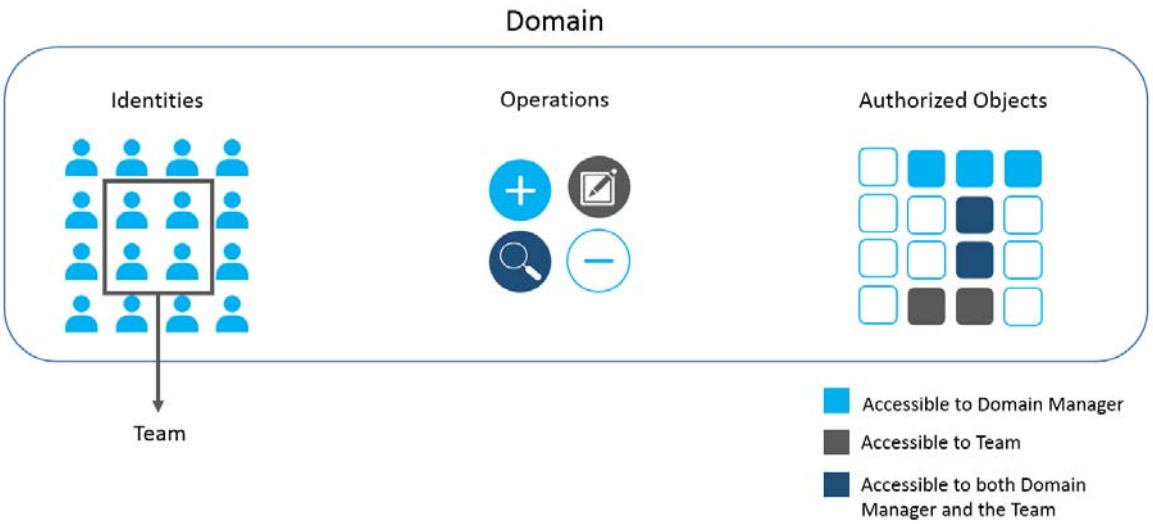
Domain Manager

A delegated administrator who has the ability to perform selected operations for a subset of authorized objects within the domain for all users.

Team Manager

A business line manager who can perform selected operations for a subset of authorized objects within the domain, but only for a designated set of users (team members).

The following diagram illustrates the scope of identity applications administrators and managers:



In this example, a Domain Administrator can perform all operations (create, modify, search, or delete) on all the authorized objects for all identities. A Domain Manager can perform the selected operations (create or search) on the selected authorized objects for all identities. A Team Manager can perform the selected operations (edit or search) on the selected objects for the set of identities that forms the team.

The identity applications do not restrict a Team Manager and a Domain Manager from accessing the same operations or authorized objects in the domain. In this example, the Team Manager and the Domain Manager have an access to search operations and some authorized objects.

Understanding Domain Administrators and Managers

The installation process initializes the Domain Administrators and Domain Managers system roles for the identity applications. However, during installation, you can specify only User Application Administrator and allow all other assignments to default to this user. After installation, you can assign accounts to the roles.

All identity applications domains have respective administrator and manager to perform the operations on all objects within the domain. You must assign a user account to each administrators.

Following are the different administrators and managers used in identity applications:

Provisioning Administrator

Required

A Domain Administrator who can perform all possible actions for all objects within the Provisioning domain.

Provisioning Manager

A Domain Manager who can perform only allowed actions for a subset of objects within the Provisioning domain.

Resource Administrator

Required

A Domain Administrator who can perform all possible actions for all objects within the Resource domain.

Resource Manager

A Domain Manager who can perform only allowed actions for a subset of objects within the Resource domain.

Role Administrator

Required

A Domain Administrator who can perform all possible actions for all objects (except for the System Roles) within the Role domain.

Role Manager

A Domain Manager who can perform only allowed actions for a subset of objects within the Role domain.

Security Administrator

Required

A Domain Administrator who can perform all possible actions for all objects within the Security domain. The Security domain allows the Security Administrator to configure access permissions for all objects in all domains within the Roles Based Provisioning Module.

The Security Administrator can configure s, and also assign domain administrators, delegated administrators, and other Security Administrators.

NOTE: For testing purposes, NetIQ does not lock down the security model in Standard Edition. Therefore, the Security Administrator is able to assign all domain administrators, delegated administrators, and also other Security Administrators. However, the use of these advanced features is not supported in production. In production environments, all administrator assignments are restricted by licensing. NetIQ collects monitoring data in the audit database to ensure that production environments comply. Furthermore, NetIQ recommends that only one user be given the permissions of the Security Administrator.

Team Manager

A user designated for a team who can perform provisioning requests and approval tasks associated with the team. Team manager is allowed to request or revoke permissions of team members. An administrator can configure a Team Manager capabilities to delegate tasks for team members.

Although a team might match a group that exists in the user directory, teams are not the same thing as groups. A group or a member of a group cannot perform team capabilities except when assigned to a team. For more information, see Managing Users in *NetIQ Identity Manager - User's Guide to the Identity Applications*.

For more information about:

- Assigning administrators, see Chapter 7, “Assigning the Identity Applications Administrators,” on page 63.
- Managing teams and team managers, see Managing Teams in NetIQ Identity Manager - User’s Guide to the Identity Applications.

Designers

Designers use the Designer for Identity Manager to customize the identity applications for an enterprise. Designer is a tool aimed at information technology professionals such as enterprise IT developers, consultants, sales engineers, architects or system designers, and system administrators who have a strong understanding of directories, databases, and their information environment and who act in the role of a designer or architect of identity-based solutions.

To create or edit workflow objects in Designer, the user needs the following rights on the RequestDefs.AppConfig container for the specific User Application driver.

- [Entry Rights] Supervisor or Create
- [All Attribute Rights] Supervisor or Write

To initiate a workflow, the user must have Browse [Entry Rights] on the RequestDefs.AppConfig container for the specific User Application driver or individually per request definition object if you are using a delegated model.

Business Users

A business user is an *authenticated user*, such as an employee, a manager, a Helpdesk user or a delegate or proxy for an employee or manager. User Application Administrator has permissions to enable capabilities for a user to perform on identity applications. For more information about user capabilities and functions, see Accessing the Identity Applications in *NetIQ Identity Manager - User’s Guide to the Identity Applications*.

Following are the users who has a special capabilities apart from their general capabilities:

Delegate user

A user to whom one or more specific tasks are delegated appropriate to the user’s rights. For example, a team manager can delegate certain tasks to a team member who has required permissions to do the delegated tasks. A Delegate user can view delegation assignments and act on those assignments.

Proxy user

A user who acts in the role of another user for a specific period assuming assignee’s identity. All of the rights of the original user apply to the proxy. The tasks owned by assignee does not carry to the proxy user. For example, a team manager assigns someone from the team as a proxy, the proxy user can assume the team manager’s role and act on the team manager’s tasks. A proxy user can view proxy assignments and act on those assignments.

Helpdesk user

A user is a part of helpdesk for a particular client. This user can assist the client users on their helpdesk tickets and resolve those tickets. A Helpdesk user can perform the following tasks with appropriate Permissions:

- Reassign an approval request that is unattended for a long time
- Browse all tasks or filter tasks for a selected user
- Request permissions on behalf of other users

3 Understanding the Functionality of the Identity Applications

Identity is the foundation of the identity applications. The applications use identity as the basis for authorizing users' access to systems, applications, and databases. Each user's unique identifier—and each user's roles—have specific access rights to identity data. For example, users who are identified as managers can access salary information about their direct reports, but not about other employees in their organization.

- "Enabling Self-Service Activities for Users" on page 35
- "Providing Permissions to Users" on page 36
- "Ensuring Permission Assignments Comply with Your Standards" on page 38
- "Design and Configuration Tools" on page 38

NOTE: The identity applications comprise an application and not a framework. The Identity Manager documentation provide instructions for modifying the applications. Modifications to areas not outlined within the product documentation are not supported.

To know more on the capabilities of Identity Manager Dashboard, watch the following videos:

Part-1

 http://www.youtube.com/watch?v=PrKa_gv5-0A

Part-2

 http://www.youtube.com/watch?v=CwjxgT5ysT0M

Enabling Self-Service Activities for Users

The identity applications provide your users a convenient way to view and work with their identity information. The activities that does not need any approval or accesses to create, modify, or delete their information in identity applications are known as self-service activities. For example, changing password which does not need any approval from the user's manager.

Identity Manager Dashboard allows your users to control their self-service activities. Following are the pages that help your users to perform their self-service activities:

Application

Lists all the applications with which user is associated.

Tasks

Shows all the user tasks that are pending for an action.

Access

Allows user to view permissions or request for permissions. User can also view **Request History** to find the status of requested permissions.

People

Allows user to view other users or groups in the system and other user's **Organization Chart**. This helps them to visualize how those users and groups are related.

Providing Permissions to Users

Permissions represent the accounts, roles, and resources that apply to users. Your organization might automatically assign permissions or users might need to request them. For example, a user might receive a computer as part of the job, but then need to request access to a specific software application. Users request permissions through the Dashboard. Some requests require approval from a single individual; others require approval from several individuals. In some instances, a request can be fulfilled without any approvals.

Following are the different ways of providing permissions to your users with an appropriate Administrator rights:

- **Assigning permissions directly to users:** You can assign a resource or a role to any user in the system.

  To assign roles, go to **Administration > Roles** and select the role that you want to assign. For more information, see "Assigning Roles to Users" on page 157.

  To assign resources, go to **Administration > Resources** and select the resource that you want to assign. For more information, see "Assigning Resource to Users" on page 167.

- **Approving user requests:** When a user requests for any permission, based on the approval/ revocation process defined for the requested permission, a corresponding task appears in the tasks list of approvers. If you are one of those approvers, you can approve the request that allows the user to use the requested permission.

  For more information about approval or revocation process, see "Changing the Approval or Revocation Process" on page 166.

  If user requests for a role that conflicts with the current role, SoD policy applied to the conflicting role must be resolved. This invokes the SoD approval flow, if any. Based on the SoD approval flow, SoD approvers see the corresponding task in their tasks list. On approving this task they can allow user to use the requested permission. For more information, see Chapter 17, "Separation of Duties Constraints," on page 175.

- **Provisioning based on workflow:** A process that coordinates the approval or revocation of a request for permissions is called as workflow. Each workflow can have automatic or manual triggers and can include email notifications.

  Workflows take into account the methods required for approving and revoking a role or resource. For example, the SAP software application might require two levels of approval: first from the user's manager and second from the resource manager for the application.

  For more information, see "Understanding Workflow-Based Provisioning" on page 37.

Understanding Workflow-Based Provisioning

Workflow-based provisioning allows you to initiate workflow processes to manage the approval and revocation of user access to your organization's secure systems.

Identity Manager Dashboard allows users to make provisioning requests (**Access > Request**). When a provisioning request requires approval from one or more individuals in an organization, the request starts one or more workflows. The workflows coordinate 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.

By default, the **New Requests** page does not display any provisioning requests. To configure a provisioning request, a designer familiar with your business needs creates a *provisioning request definition*, which binds the resource to a workflow.

The designer can configure workflows that proceed in one of the following ways:

- *Sequential* fashion, with each approval step being performed in order
- *Parallel* fashion, which allows more than one user to act on a workflow task concurrently

Identity Manager provides a set of Eclipse-based tools for designing the data and the flow of control within the workflows. In addition, Identity Manager provides a set of Web-based tools that allow users to view existing provisioning requests and manage workflows that are in process. For more information, see "Design Constraints" on page 53

The Provisioning Administrator is responsible for managing the workflow-based provisioning features of identity applications. For more information, see Chapter 2, "Types of User Categories in Identity Applications," on page 29.

Understanding a Client Helpdesk

The Dashboard includes Helpdesk to help users troubleshoot any issues while performing their tasks in Identity Manager.

Some of the tasks that Helpdesk can perform are:

- Reassign an approval request that is unattended for a long time
- Browse all tasks or filter tasks for a selected user
- Request permissions on behalf of other users

Users can contact Helpdesk by using the Helpdesk email ID, contact number, or raise a Helpdesk ticket. When a client user raises a ticket, the Helpdesk user receives a notification on the Dashboard. By default, Helpdesk is not configured. Administrators need to configure Helpdesk for the clients configured in the system.

After setting up a Helpdesk, the administrator can customize the Helpdesk information for the clients from the Dashboard client settings. To set up a Helpdesk and configure the Helpdesk information, see "Configuring a Client Helpdesk" on page 136.

Ensuring Permission Assignments Comply with Your Standards

Compliance is the process of ensuring that an organization conforms to relevant business laws and regulations. One of the key elements of compliance is attestation, which provides a method for organizations to verify that personnel are fully aware of organizational policies and are taking steps to comply with these policies. By requesting that employees or administrators regularly attest to the accuracy of data, management ensures that personnel information such as user profiles, role assignments, and approved SoD exceptions are up-to-date and in compliance.

NOTE: For compliance and attestation processes, we recommend using **NetIQ Identity Governance** (formerly Access Review) instead of the identity applications. Identity Governance enables administrators and managers to easily collect all user and access information in one central location and certify that each user has only the level of access that they need to do their job. Following the principle of least privilege, Access Review helps you ensure that your users have focused access to those applications and resources that they use and cannot access resources that they do not need to access. You can review all permissions assigned to your employees, either individually or as a group, and decide whether those permission assignments are appropriate. For more information, see the NetIQ Identity Access Governance documentation.

Design and Configuration Tools

The various administrators can use the following tools to design and configure the Identity Applications.

Table 3-1 *Tools for Designing and Configuring the User Application*

Tool	Purpose
Designer for Identity Manager	A powerful, graphical toolset for configuring and deploying Identity Manager. The following plug-ins are designed to help you configure the User Application: ◆ Directory Abstraction Layer editor: Lets you define the Identity Vault objects for your User Application. ◆ Provisioning Request Definition editor: Lets you create workflows for provisioning request definitions. Also allows you to customize the forms by which users make and approve requests and email templates. ◆ User Application driverProvisioning view: Lets you import, export, deploy, and migrate directory abstraction layer and provisioning requests to the . ◆ Role editor: Lets you create and configure roles for use within the User Application. ◆ Resource editor: Lets you create and configure resources for use within the User Application. For more information, see the *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.

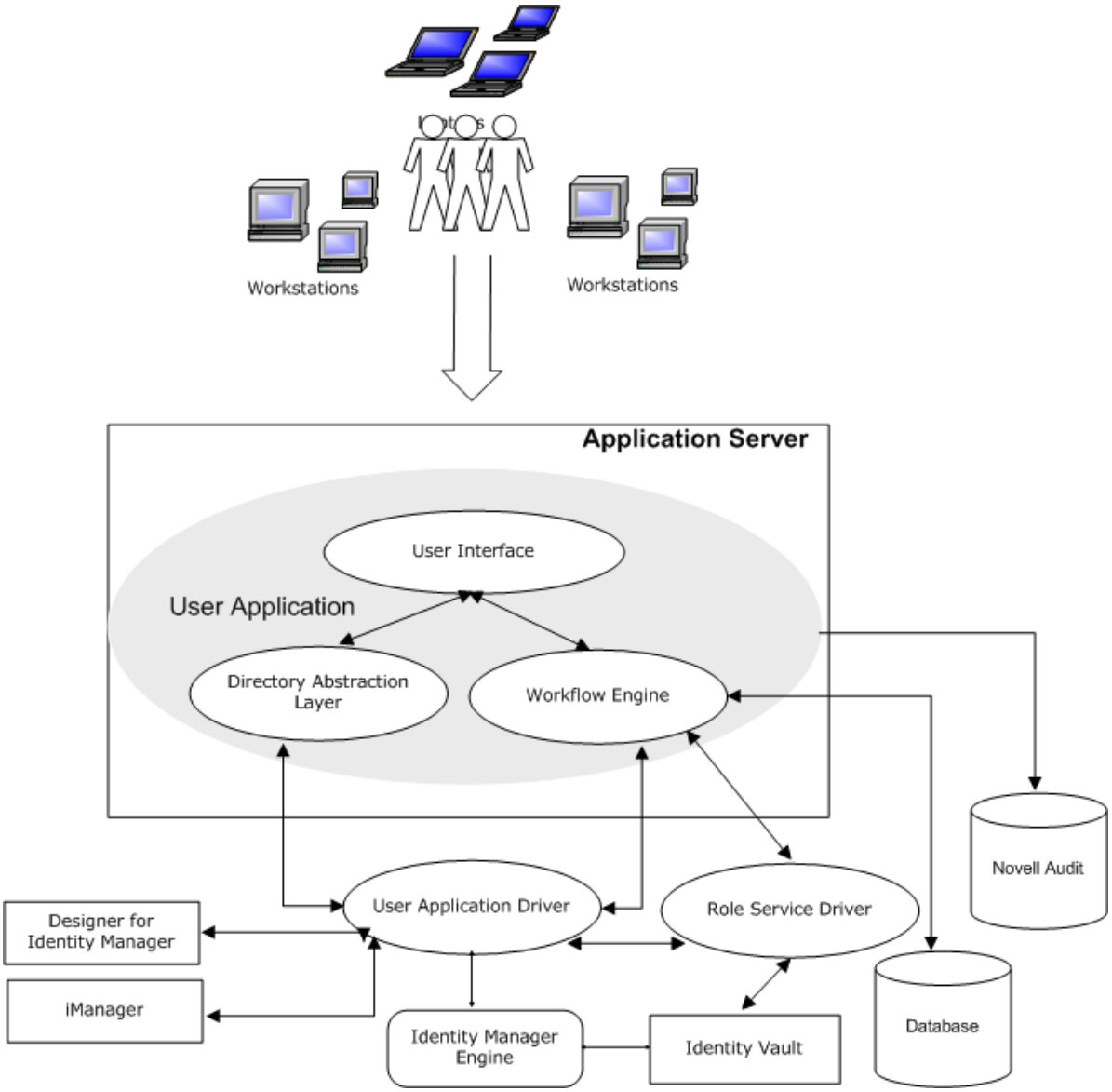
Tool	Purpose
iManager	A Web-based administration console. The following plug-ins are designed to help you configure and administer the User Application: ◆ Provisioning Request Configuration plug-in: Provides a read-only view of provisioning request definitions d through Designer and allows you to mark them active or inactive. ◆ Workflow Administration plug-in: Provides a browser-based interface that lets you view the status of workflow processes, reassign activities within a workflow, or terminate a workflow in the event that it is stopped and cannot be restarted. ◆ Provisioning Team plug-in: Not supported with this release of the Roles Based Provisioning Module. The **Team Configuration** user interface on the **Administration** of the User Application replaces this iManager tool. ◆ Provisioning Team Request plug-in: Not supported with this release of the Roles Based Provisioning Module. The **Team Configuration** user interface on the **Administration** of the User Application replaces this iManager tool. For more information, see Part IV, “Configuring and Managing Provisioning Workflows,” on page 215
Identity Manager Dashboard	A Web-based administration console that allows you to configure, manage, and customize the identity applications. ◆ **Administration:** Allows you to create roles, resources, SoD, Permission Reconciliation settings. For more information, see Part III, “Identity Applications Administration,” on page 151. ◆ **Settings:** Customize and configure client and helpdesk settings. Also, enable dashboard widgets to your users. For more information, see “Configuring Client Settings Mode” on page 130.

4 Understanding the Back-end Functions for the Identity Applications

The identity applications rely on a number of components acting together:

- "User Interfaces" on page 42
- "Directory Abstraction Layer" on page 43
- "Workflow Engine" on page 43
- "SOAP Endpoints" on page 43
- "Application Server" on page 44
- "Database" on page 45
- "User Application Driver" on page 45
- "Role and Resource Service Driver" on page 46
- "Multi-Threaded Role and Resource Service Driver" on page 47
- "Designer for Identity Manager" on page 49
- "iManager" on page 49
- "Identity Manager Engine" on page 49
- "Identity Vault" on page 49

The following figure shows how these components fit into the overall architecture of the identity applications.



User Interfaces

Users interact with the identity applications through a Web browser, based on Java applications and a Tomcat application server. Identity Manager Dashboard serves as a primary portal to the identity applications. The application server on which the applications run provides various services to the application as a whole, such as scalability through clustering, database access via JDBC, and support for certificate-based security.

Directory Abstraction Layer

The directory abstraction layer provides a logical view of the Identity Vault data. You define a set of entities and their related attributes based on the Identity Vault objects that you want users to view, modify, or delete in the identity applications. The Directory Abstraction layer:

- Performs all of the LDAP queries against the Identity Vault. This isolates presentation-layer logic from the Identity Vault, so that all requests for identity data go through the directory abstraction layer.
- Checks constraints and access control on data requests made with the identity applications.
- Caches runtime configuration and entity-definition data obtained from the Identity Vault. See "Configuring Caching and Cluster Settings" on page 193.

You use the directory abstraction layer editor plug-in (available in Designer for Identity Manager) to define the structure of the directory abstraction layer data definitions. To learn more, see the section on the directory abstraction layer editor in the Configuring the Directory Abstraction Layer in the *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.

Workflow Engine

The Workflow Engine is a set of Java executables responsible for managing and executing steps in an administrator-defined workflow and keeping track of state information (which is persisted in a database). When the necessary approvals have been given, the Provisioning System provisions the resource as requested.

During the course of workflow execution, the Workflow Engine can send one or more email messages to notify users of changes in the state of the workflow. In addition, it can send email messages to notify users when updates have been made to proxy, delegate, and availability settings.

You can edit an email template in the Designer for Identity Manager or in iManager and then use this template for email notifications. At runtime, the Workflow Engine retrieves the template from the directory and replaces tags with dynamic text suitable for the notification.

Additional details about the Workflow Engine, including how to configure and manage provisioning workflows, are in Part IV, "Configuring and Managing Provisioning Workflows," on page 215.

SOAP Endpoints

The identity applications provide the following SOAP endpoints to allow third-party software applications to take advantage of identity applications services:

SOAP Endpoint	Description
Provisioning Web Service	To support third-party access, the provisioning Workflow Engine includes a Web service endpoint. The endpoint offers all provisioning functionality (for example, allowing SOAP clients to start a new approval flow, or list currently executing flows).

SOAP Endpoint	Description
Metrics Web Service	The workflow engine also includes a Web Service for gathering workflow metrics. The addition of the Metrics Web Service to the Workflow Engine lets you monitor an approval flow process. In addition, it provides indicators the business manager can use to modify the process for optimal performance.
Notification Web Service	The Provisioning System includes an email notification facility that lets you send email messages to notify users of changes in the state of the provisioning system, as well as tasks that they need to perform. To support third-party access, the notification facility includes a Web service endpoint that lets you send an email message to one or more users.
Directory Abstraction Layer (VDX) Web Service	The directory abstraction layer provides a logical view of the Identity Vault data. To support access by third-party software applications, the directory abstraction layer includes a Web service endpoint called the VDX Web Service. This endpoint lets you access the attributes associated with entities defined in the directory abstraction layer. It also lets you perform ad hoc searches for entities and execute predefined searches called global queries.
Role Web Service	To support access by third-party software applications, the Role subsystem includes a Web service endpoint called the Role Web Service. It supports a wide range of role management and SoD management functions.

Application Server

The application server provides the runtime framework in which the identity applications, directory abstraction layer, and Workflow Engine execute. The identity applications are packaged as a Java Web Application Archives, or WAR files.The installation process enables you deploy the WAR files to the application server.

The following WAR files apply to the URL for a component of the identity applications:

- **IDMProv** for the User Application
- **idmdash** for the Dashboard
- **idmadmin** for Identity Applications Administration

When a user interacts with the *idmdash* or *idmadmin* applications, it queries the underlying *IDMProv.war* file and fetches the information for the user. *IDMProv.war* contains the REST and SOAP APIs where *idmdash* and *idmadmin* contains the information that provides the user interface.

The identity applications run on an Apache Tomcat application server, included in the installation kit. You can also use your own installation of Tomcat. For more information about the application server requirements, see “Considerations for Installing Identity Manager Components ” in the *NetIQ Identity Manager Setup Guide for Linux* or “Installing Identity Applications” in the *NetIQ Identity Manager Setup Guide for Windows*.

Database

Most user information is stored in the Identity Vault. However, the identity applications rely on a separate database to store the following information:

- Configuration data for the identity applications, such as Web page definitions and preference values
- State of a workflow

NOTE: The actual workflow definitions are stored in the Identity Applications driver in the Identity Vault.

For more information about installing and configuring database, see "Installing and Configuring Identity Manager Components" in the *NetIQ Identity Manager Setup Guide for Linux* or "Installing Identity Applications" in the *NetIQ Identity Manager Setup Guide for Windows*.

User Application Driver

The User Application driver is responsible for:

- Storing application-specific environment configuration data.
- Notifying the directory abstraction layer when important data values change in the Identity Vault. This causes the directory abstraction layer to update its cache.

You can configure the User Application driver to:

- Allow events in the Identity Vault to trigger workflows.
- Communicate the success or failure of a workflow's provisioning activity back to the identity applications database, which allows users to view the final status of their requests.
- Start workflows automatically in response to changes of attribute values in the Identity Vault.

The User Application driver is both a runtime component and a storage wrapper for directory objects (comprising the runtime artifacts of the identity applications).

Artifacts	Description
Driver Set Object	Every Identity Manager installation requires that drivers be grouped into driver sets. Only one driver set can be active at a time (on a given directory server). The drivers within that set can be toggled on or off individually without affecting the driver set as a whole. The User Application driver like any other Identity Manager driver, must exist inside a driver set. The driver set is not automatically created by the User Application; you must create one, then create the User Application driver within it.
User Application	The User Application driver object is the container for a variety of artifacts. The User Application driver implements Publisher and Subscriber channel objects and policies. The Publisher channel is not used by the User Application but is available for custom use cases.

Artifacts	Description
App Config Object	The AppConfig object is a container for the following User Application configuration objects. ◆ **RequestDefs:** Container for Provisioning Request Definitions. The definitions stored here (as XML) represent the classes of requests that end users with appropriate rights can instantiate via the User Application. ◆ **WorkflowDefs:** Container for Workflow objects, including design-time descriptions plus any template or unused flows. ◆ **ResourceDefs:** Container for Provisioned Resource definitions, including design-time descriptions plus any templates or unused targets. ◆ **ServiceDefs:** Container for Service Definition objects, which wrap Web Services called by workflows. ◆ **DirectoryModel:** Directory abstraction layer objects that represent different types of content of the Identity Vault that can be exposed in the User Application. ◆ **AppDefs:** Container for configuration objects that initialize the runtime environment, such as cache configuration information and email notification properties. ◆ **ProxyDefs:** Container for proxy definitions. ◆ **DelegateeDefs:** Container for delegate definitions.

Role and Resource Service Driver

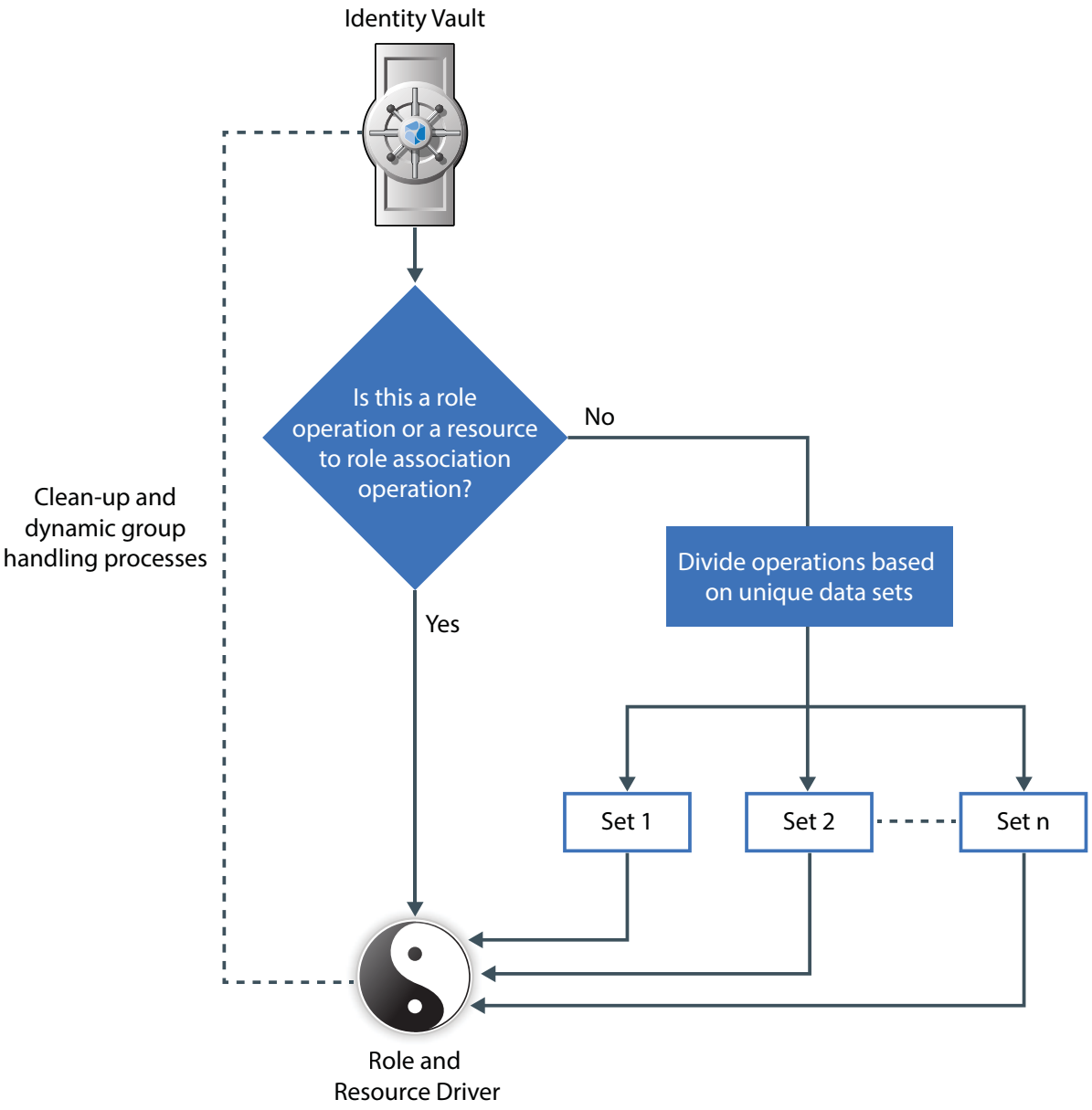
The identity applications use the Role and Resource Service Driver to manage back-end processing of resources:

- Starts an SoD workflow and waits for approvals in situations where a role request requires an SoD workflow.
- Starts a role assignment workflow and waits for approvals in situations where a role request requires a workflow.
- Adds users to and remove users from roles. To do this, the Role and Resource Service driver:
  - Waits for a start date before making assignments
  - Terminates a role assignment when the end date is reached
- Adds and removes higher-level and lower-level role relationships.
- Adds and removes role assignments for groups.
- Adds and removes role assignments for containers.
- Maintains all role membership information for indirect role assignments, including:
  - Role assignments acquired through role relationships
  - Role assignments that result from membership in groups
  - Role assignments that result from membership in containers
- Grants and revokes entitlements to and from users according to their role memberships.

- Maintains additional reporting information that is associated with each role assignment.
- Maintains additional reporting information on objects in eDirectory, such as:
  - Approval information
  - Where indirect assignments come from
  - Where entitlements come from
- Logs events to an auditing service.
- Cleans up processed requests after a user-specified amount of time.
- Recalculates role assignments based on dynamic and nested groups on a polled basis.

Multi-Threaded Role and Resource Service Driver

A traditional Role and Resource Service driver contains a single thread of control for managing all driver events, such as role and resource requests. A multi-threaded driver works on multiple unique data sets so that requests for different unique data sets can be simultaneously processed. To accomplish this, the driver uses worker threads. The main thread listens to the incoming requests and is responsible for passing on the requests to the appropriate worker threads. Each worker thread can execute independently on a unique identity data set.



A unique identity data set comprises of data, such as users, groups, and containers, that is different based on certain attributes. When different unique identity data sets are configured in your environment, the driver uses worker threads to accomplish the tasks belonging to different disjoint sets at the same time. In addition, it remains responsive while doing lengthy or memory intensive operations. For example, while recalculating resources for a particular user, a multi-threaded driver can perform role or resource assignments on other users belonging to a different data set.

IMPORTANT: When the driver receives role and resource associations and role processing events, such as creation or deletion of roles, it starts processing these events only after completing the processing of all the events that are already submitted to the worker threads. This prevents any adverse impact on other operations. Similarly, it does not take up any new events until it finishes processing the current event and starts working in a single-thread mode.

By default, the driver is enabled for multi-threaded service. To configure the driver, see Chapter 12, “Configuring a Multi-Threaded Role and Resource Service Driver,” on page 143.

Designer for Identity Manager

Designer for Identity Manager provides a set of plug-ins that you can use to define the directory abstraction layer objects and provisioning requests and their associated workflows. For more information, see “Design and Configuration Tools” on page 38.

iManager

iManager provides a set of plug-ins you can use to view provisioning requests and manage their associated workflows. For more information, see “Design and Configuration Tools” on page 38.

Identity Manager Engine

The Identity Manager engine provides the runtime framework that monitors events in the Identity Vault and connected systems. It enforces policies and routes data to and from the Identity Vault. The Identity Manager User Application is a connected system. Communication between the Identity Vault, the directory abstraction layer, and the Workflow Engine occurs through the User Application driver.

Identity Vault

The Identity Vault is the repository for:

- User data
- Other identity data
- Identity Manager driver set
- User Application driver

The User Application relies on various Identity Vault objects, so it is necessary to extend the eDirectory schema to accommodate the custom LDAP objects and attributes required by the User Application.

The identity applications schema extension occurs automatically as part of the install. The custom objects and attributes are populated with default values after the User Application driver is installed and activated.

II Preparing the Identity Applications for Use

This section helps you set up your production environment for the identity applications.

- Chapter 5, “Understanding the Design Needs,” on page 53
- Chapter 6, “Configuring Security in the Identity Applications,” on page 55
- Chapter 7, “Assigning the Identity Applications Administrators,” on page 63
- Chapter 8, “Setting Up Logging in the Identity Applications,” on page 67
- Chapter 9, “Tuning the Performance of the Applications,” on page 97
- Chapter 10, “Customizing the Identity Applications for Your Enterprise,” on page 111
- Chapter 11, “Setting Up the Dashboard for Identity Applications,” on page 141
- Chapter 12, “Configuring a Multi-Threaded Role and Resource Service Driver,” on page 143
- Chapter 13, “Configuring Identity Applications Clustering and Permission Clustering,” on page 147

For more information about installing the identity applications, see “Considerations for Installing Identity Manager Components ” in the *NetIQ Identity Manager Setup Guide for Linux* or “Planning to Install the Identity Applications” in the *NetIQ Identity Manager Setup Guide for Windows*.

5 Understanding the Design Needs

Each major subsystem can have many instances and many ways of connecting. Not every possible layout is supported. This section provides information about design constraints and using a high-availability environment.

- "Design Constraints" on page 53
- "High Availability Design" on page 54

Design Constraints

In general, you install the Identity Manager components on specific servers, as described in the *NetIQ Identity Manager Setup Guide for Linux* or *NetIQ Identity Manager Setup Guide for Windows*. When configuring the identity applications, you also need to consider the following architectural constraints:

One user container per identity applications instance

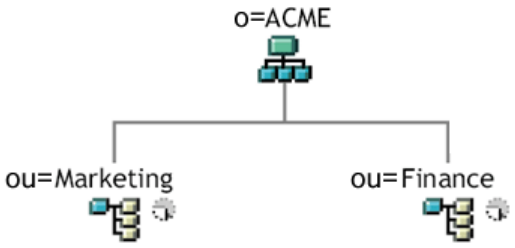
No instance of the identity applications can service, such as search, query, or add users to, more than one user container. Also, a user container association with the applications is meant to be permanent.

One User Application driver per identity applications instance

No User Application driver can be associated with more than one instance of the identity applications, except when the applications are installed on sister nodes of the same cluster. In other words, Identity Manager does not support a one-to-many mapping of drivers to identity applications instances.

The first constraint enforces a high degree of encapsulation in User Application design. Suppose you have the following organizational structure:

Figure 5-1 *Sample Organizational Structure*



During installation of the identity applications, you are asked to specify the top-level user container that your installation looks for in the Identity Vault. In this case, you could specify `ou=Marketing,o=ACME` or (alternatively) `ou=Finance,o=ACME` identity applications. You cannot specify both. All searches and queries (and administrator logins) for the are connected to whichever container you specify.

NOTE: In theory, you could specify a scope of `o=ACME` in order to encompass Marketing and Finance. But in a large organization, with potentially many `ou` containers (rather than just two relating to Marketing and Finance), this is not likely to be practical.

It is possible to create two independent installations of the identity applications that share no resources in common: one for Marketing and another for Finance. Each installation would have its own database and its own appropriately configured User Application driver. Also, each would be administered separately, possibly having unique User Application drivers.

If you truly need to place Marketing and Finance within the same scope for one installation, you can consider one of the following tactics:

- Insert a new container object (for example, `ou=MarketingAndFinance`) in the hierarchy, above the two sibling nodes; then point to the new container as the scope root.
- Create a filtered replica (a special type of eDirectory tree) that combines the needed parts of the original ACME tree, and point the identity applications at the replica's `root` container. For more information about filtered replicas, see the *eDirectory Administration Guide*.

If you have questions about a particular system layout, contact your NetIQ representative for assistance or advice. For more information about design constraints, see "Planning Overview" and "Considerations for Installing Identity Manager Components " in the *NetIQ Identity Manager Setup Guide for Linux* and "Planning to Install the Identity Applications" in the *NetIQ Identity Manager Setup Guide for Windows*.

High Availability Design

You can provide high availability of the identity applications by installing in a cluster. Set up a cluster so that each node runs one instance of the identity applications. The instances are all coequals (peers). The support automatic failover, where an interrupted workflow can resume after the loss of a cluster node.

For more information about using a clustered environment, see the following sections:

- "Planning Overview" in the *NetIQ Identity Manager Setup Guide for Linux*.
- "Planning Overview" in the *NetIQ Identity Manager Setup Guide for Windows*.
- "High Availability Design" on page 54
- "Configuring Caching and Cluster Settings" on page 193

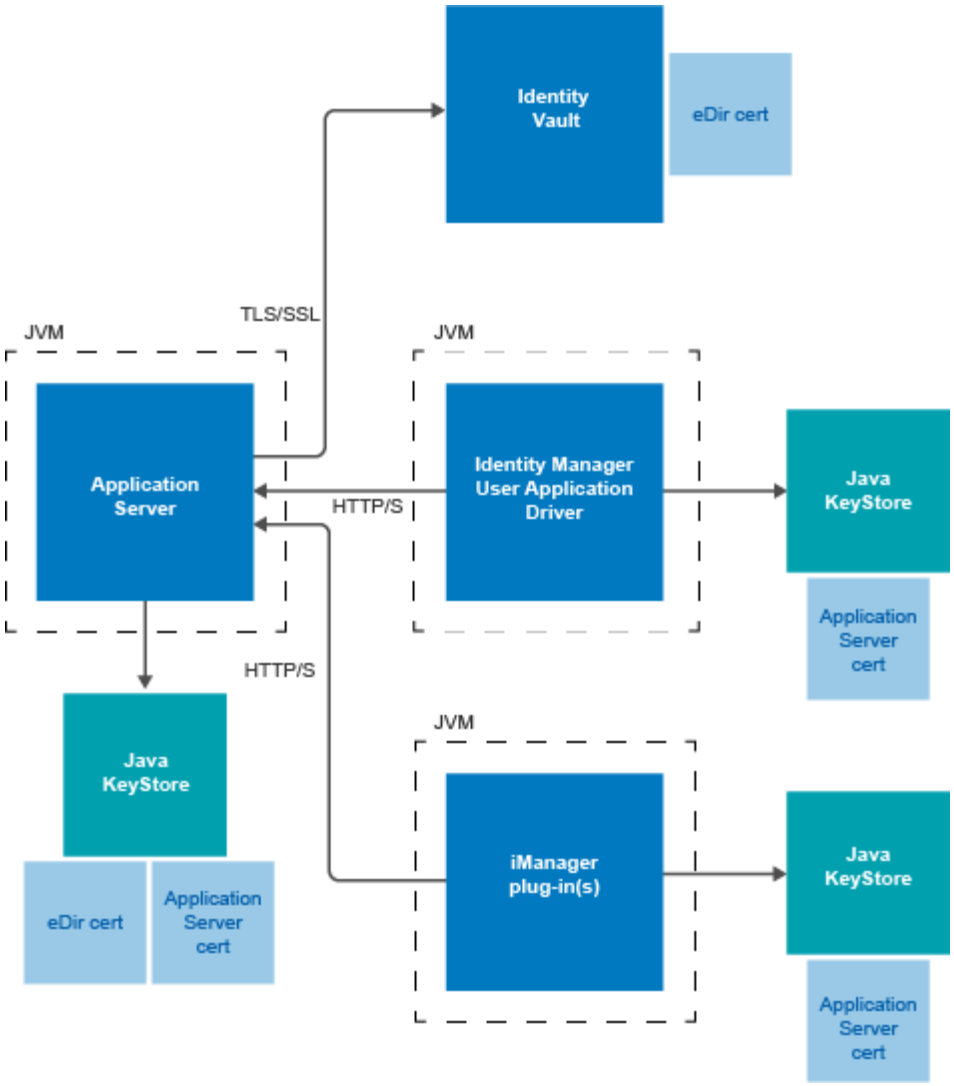
6 Configuring Security in the Identity Applications

Moving from pre-production to production usually involves hardening the security aspects of the system. In sandbox testing, you might use regular HTTP to connect the User Application driver to the application server, or you might use a self-signed certificate (as a temporary measure) for driver/app-server communication. In production, on the other hand, you probably use secure connections, with server authentication based on your company's Verisign* (or other trusted provider) certificate.

- "Understanding Security in the Identity Applications Environment" on page 55
- "Using Secure Sockets for User Application Connections to the Identity Vault" on page 57
- "Enabling SSL for User Access" on page 58
- "Enabling SOAP Security" on page 58
- "Enabling Authentication" on page 59
- "Encrypting Sensitive Identity Applications Data" on page 59
- "Preventing XSS Attacks" on page 60
- "Modifying Trustee Rights" on page 60
- "Updating a Password for a Database User on Tomcat" on page 62

Understanding Security in the Identity Applications Environment

It is typical for X.509 certificates to be used in a variety of places in the identity applications environment, as shown in the following diagram.



All communication between the identity applications and the Identity Vault is secure, using Transport Layer Security, by default. The installation of the Identity Vault (eDirectory) certificate into the Tomcat application server keystore is done automatically during installation time. Unless you specify otherwise, the installer places a copy of the eDirectory certificate in the JRE's default *cacerts* store. For more information, see the *NetIQ Identity Manager Setup Guide for Linux* or *NetIQ Identity Manager Setup Guide for Windows*.

The server certificate needs to be in several places, if communications are to be secure, as shown in the diagram. Different setup steps might be needed depending on whether you intend to use a self-signed certificate in the various places in the diagram shown with a *Application Server cert* box, or you intend to use a certificate issued by a trusted certificate authority (CA) such as Verisign.

Using Secure Sockets for User Application Connections to the Identity Vault

By default, secure sockets are used for communication between the User Application server and the Identity Vault. However, in some environments, not all communication needs to be secured. For example, if the User Application and Identity Vault servers are on an isolated network, and the only ports available to the outside are the HTTP ports, it might be acceptable for some communication between the two servers to be accomplished using non-secure sockets. Some aspects of the application will *always* use a secure connection (for example, a user changing a password) even though the setting might indicate that secure connections are not required. Turning off secure connections, especially for user connections, can greatly increase performance and scalability. If, in a particular environment, there are many concurrent logins, and communication between the User Application server and the Identity Vault server have been secured using the network setup, then turning off the secure connection for user connections greatly increase the number of concurrent logins that can be processed. We recommend that this option be used only when there is actual evidence of scaling or performance problems in the environment, and adding additional eDirectory servers is not an option.

Additionally, secure connections can be turned off for administrative connections. These connections are used for general queries on the Identity Vault server that do not require user credentials. These connections are pooled and used round-robin. The bind over a secure connection is only done once at application startup (or possibly again later on if the connection becomes unresponsive) and so does not represent the scalability issues that can arise with the user connections. However, the time it takes to encrypt and decrypt the data at both ends does add overhead. We recommend that the default setting be used, unless there is a need to gain extra performance.

Secure communications for administrative and user connections must be disabled in both the User Application and in iManager.

Disabling Secure Communications Using the Configuration Update Utility

To disable the secure administrative and user connections in the User Application:

1. Run the configupdate script, located in the User Application directory, as follows:
   - Linux: Type the following to run `configupdate.sh`:

     ```
     ./configupdate.sh
     ```

   - Windows: Run `configupdate.bat`

   Launches Configuration Update utility.
2. Deselect **Secure Admin Connection** and **Secure User Connection**.
3. Click **OK**.

Disabling Secure Communications Using iManager

To disable the requirement for secure LDAP (LDAPS) connections for administrative and user connections to eDirectory using iManager:

1. Log into your eDirectory tree.
2. Navigate to the **LDAP** group object and display its properties.
3. Click **General**.
4. Deselect **Require TLS for Simple Binds with Password**.

NOTE: In a multi-server eDirectory tree, disabling TLS on the LDAP group removes the TLS requirement from all servers. If you want mixed TLS requirements for each individual server in your tree, you must enable the TLS requirement on each server.

Enabling SSL for User Access

The identity applications use HTML forms for authentication. As a result, user credentials are exposed during log in. We strongly recommend that you enable SSL to protect sensitive information. For more information, see "Checklist for Ensuring SSL Connections" on page 581.

Enabling SOAP Security

1. In `IDMProv.war`, find the `web.xml` file and open it in a text editor.
2. At the bottom of the file, uncomment the following section:

```
<security-constraint>
        <web-resource-collection>
                <web-resource-name>IDMProv</web-resource-name>
                <description>IDM Provisioning Edition</description>
                <url-pattern>/*</url-pattern>
                <http-method>POST</http-method>
                <http-method>GET</http-method>
                </web-resource-collection>
        <user-data-constraint>
                <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
</security-constraint>
```

3. Save the file and archive, then restart Tomcat.

Enabling Authentication

Enabling Mutual Authentication

The Identity Manager User Application does not support client certificate-based authentication out of the box. That functionality can be obtained, however, by using NetIQ Access Manager. See your NetIQ representative for more information. See also "Enabling Third-Party Authentication and Single Sign-On" on page 59.

Enabling Third-Party Authentication and Single Sign-On

You can configure Identity Manager to work with NetIQ Access Manager using SAML 2.0 authentication. This capability enables using a non-password-based technology to log in to the identity applications through Access Manager. For example, users can log in through a user (client) certificate, such as from a smart card.

Access Manager interacts with One SSO Provider (OSP) in Identity Manager to map the user to a DN in the Identity Vault. When a user logs in to the identity applications through Access Manager, Access Manager can inject a SAML assertion (with the user's DN as the identifier) into an HTTP header and forwards the request to the identity applications. The identity applications use Proxied Control Authorization to establish the LDAP connection with the Identity Vault, see NetIQ eDirectory Administration Guide. For information on configuring Access Manager to support this capability, refer to the Access Manager documentation.

Accessory portlets that allow single sign-on authentication based on passwords do not support single sign-on when SAML assertions are used for identity application authentication.

For more information about configuring Identity Manager to work with Access Manager, see "Using SAML Authentication for Single Sign-on" on page 546.

Encrypting Sensitive Identity Applications Data

Any sensitive information associated with the User Application that is stored persistently is encrypted by using the symmetric algorithm AES-128. The master key itself is protected by password-based cryptography using PBEWithSHA1AndDESede. The password is never persisted or stored out of memory.

Information that is encrypted includes (but is not limited to):

- LDAP administrator user password
- LDAP guest user password
- DSS trusted CA keystore password
- DSS signature key keystore password
- DSS signature key entry password

However, in a cluster environment, if session failover is enabled, some sensitive data (for example, a login-password for single sign-on) in the user session can be transferred on the network during session replication. This can expose sensitive data to network sniffers. To protect this sensitive data, do one of the following:

- Enable encryption for JGroups. For information about enabling JGroups encryption, see JGroups Encrypt (http://wiki.jboss.org/wiki/Wiki.jsp?page=JGroupsENCRYPT).
- Make sure that the cluster is behind a firewall.

Preventing XSS Attacks

The User Application supports the concept of XSS (Cross-Site Scripting) blacklists to help you to prevent scripting attacks. The XSS blacklists prevent XSS injection in the free text input fields within the Detail portlet, approval flow, and role assignments pages within the application.

The User Application provides default values for two blacklists, one for the Detail Portlet, and another for the workflow system (which handles the approval flow and role assignments pages). However, you can customize the blacklists to suit the requirements of your environment.

To customize the either of the blacklists, you need to enter the words or characters you want to block in the `ism-configuration.properties` file. In Tomcat, you can find this file in the `<tomcat_home>/conf` folder.

To modify the blacklist for the Detail portlet, open `<tomcat_home>/conf/ism-configuration.properties` and find the `com.novell.xss.blacklist.detailportlet` property:

```
com.novell.xss.blacklist.detailportlet = \",<
```

By default, the following characters are not allowed: \",<

To modify the blacklist for the approval flow and role assignments pages, locate the `com.novell.xss.blacklist.workflow` property:

```
com.novell.xss.blacklist.workflow = <
```

By default, < character is not allowed.

If you decide to customize the blacklists, be careful not to remove the default values. If you remove these values, you will make the lists less restricted, and therefore increase the risk of XSS attacks.

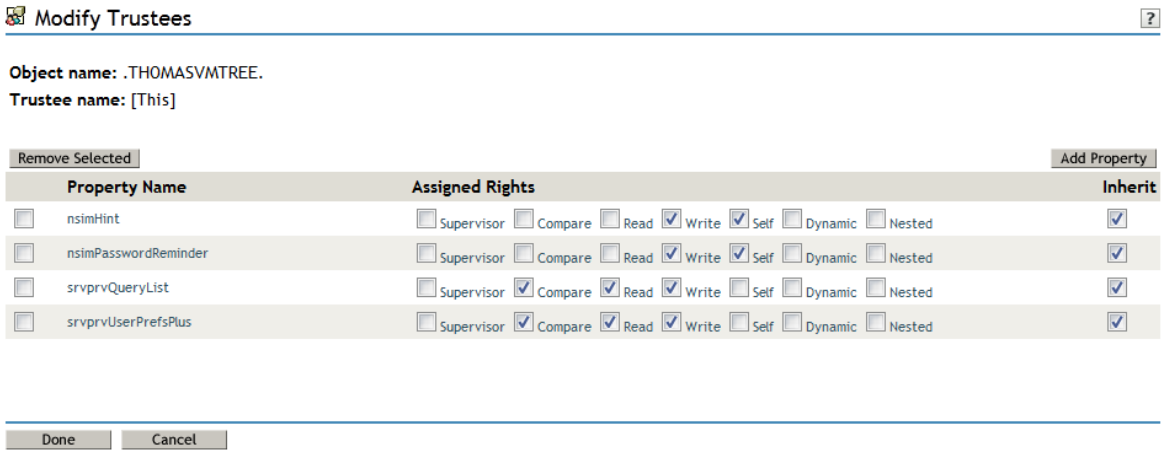
Modifying Trustee Rights

To perform tasks within the identity applications, users must have specific trustee rights.

Modifying the Trustee Rights for User Preferences

To allow user preferences to be saved, the administrator must ensure that the permissions on the srvprvUserPrefsPlus and srvprvQueryList attributes are set so that the user is able to write to these attributes. The necessary rights should be set for [This] at the tree root level, since [This] is a special

alias to the object itself, causing only the user to have rights to modify its own preferences. To set the proper permissions, the administrator needs to modify the trustees for these attributes in iManager, as shown below:



The srvprvUserPrefsPlus property has no space limitations, so it can save a large amount of user preference information. If you have used the srvprfUserPrefs property in a previous release, this property will be migrated to srvprvUserPrefsPlus the first time a user saves new preferences in the User Application.

Modifying the Trustee Rights for a Provisioning Request Definition

To view the details and comments associated with a task in the **Task Notifications** section of the Work Dashboard tab, the Domain Administrator or Delegated Administrator must have the proper rights to the provisioning request definition. In particular, the user must have the nrfAccessMgrTaskAddressee right to the provisioning request definition, with write access enabled. To set the proper permissions, the administrator needs to modify the trustees for the provisioning request definition, as described below:

1. Log into iManager as an administrator.
2. Select **Modify Trustee** from the **Rights** left-navigation menu.
3. Browse to the provisioning request definition.
4. If necessary, click **Add Trustee** to add the user.
5. Click on the **Assign rights** link.

   Notice that nrfAccessMgrTaskAddressee is not listed with the write permission checked, which means that the user does not have the proper rights for the provisioning request definitiion.
6. Click the **Add Property** button.
7. Check the check box for **Show all properties in schema**.
8. Select nrfAccessMgrTaskAddresss.
9. Check the **write** checkbox for **Assigned Rights**.

10 Click **Done**.

11 Click **OK**.

Restricting a User from Viewing Provisioning Request Definitions and Roles in Identity Applications

Using Designer or iManager, you can restrict users from viewing roles or provisioning request definitions (PRDs) in Identity Applications for which they are not assigned as trustees. Trustee assignments can be configured for users, groups, and containers.

- To restrict a user from viewing roles, control the users' permission to the RoleDefs container by setting the user as a trustee of individual roles in the Identity Vault.
- To restrict a user from viewing PRDs for which the user is not assigned as a trustee, modify the trustee assignments for the RequestDefs container.

Perform the following steps to modify the trustee assignments for the RoleDefs container in iManager:

1 In Roles and Tasks, select **Rights > Modify Trustees**.

2 Browse to the container whose trustee list you want to modify, and then click **OK**.

This opens a list of the object's currently assigned trustees. For example, `RoleDefs.AppConfig.UserApplication Driver.driverset1.system`.

3 Add a trustee to the container by clicking **Add Trustee**, and then select **[Public]**.

4 Click **Assigned Rights** to assign the trustee's rights.

5 Deselect all options under **Assigned Rights** for the **[All Attributes Rights]** property.

6 Select **Browse** for the **[Entry Rights]** property. Ensure that all assigned rights are deselected and only **Inherit** is selected.

7 Click **Done**.

To restrict a user from viewing PRDs, repeat Steps 1-7 for the RequestDefs container.

Updating a Password for a Database User on Tomcat

Perform the following actions to update the database user's password in the database server.

1 Stop Tomcat.

2 Update the password in the database server.

3 With Java in your path, enter the following command:

```
java -jar idm/apps/tomcat/lib/idm-datasource-factory-1.2.0-uber.jar
%newpassword%
```

4 Copy the encrypted output of the password to the server.xml file.

5 Save and close the file.

6 Start Tomcat.

7 Assigning the Identity Applications Administrators

The identity applications support several types of users. To make administrative-type changes to the applications, you must be assigned to at least one of the administrator or manager roles.

- "Understanding the Administrators of the Identity Applications" on page 63
- "Changing the Default Administrator Assignments after Installation" on page 64

To assign administrators using Identity Manager Dashboard, see "Assigning Administrators in Identity Applications" on page 199.

Understanding the Administrators of the Identity Applications

The installation process initializes the Domain Administrators and Domain Managers system roles for the identity applications. However, during installation, you can specify only the User Application Administrator and allow all other assignments to default to this user. After installation, you can assign accounts to the roles.

You must assign an account to the roles that have an Administrator title.

Provisioning Administrator

Required

A Domain Administrator who can perform all possible actions for all objects within the Provisioning domain.

Provisioning Manager

A Domain Manager who can perform only allowed actions for a subset of objects within the Provisioning domain.

Resource Administrator

Required

A Domain Administrator who can perform all possible actions for all objects within the Resource domain.

Resource Manager

A Domain Manager who can perform only allowed actions for a subset of objects within the Resource domain.

Role Administrator

Required

A Domain Administrator who can perform all possible actions for all objects (except for the System Roles) within the Role domain.

Role Manager

A Domain Manager who can perform only allowed actions for a subset of objects within the Role domain.

Security Administrator

Required

A Domain Administrator who can perform all possible actions for all objects within the Security domain. The Security domain allows the Security Administrator to configure access permissions for all objects in all domains within the Roles Based Provisioning Module.

The Security Administrator can configure s, and also assign domain administrators, delegated administrators, and other Security Administrators.

NOTE: For testing purposes, NetIQ does not lock down the security model in Standard Edition. Therefore, the Security Administrator is able to assign all domain administrators, delegated administrators, and also other Security Administrators. However, the use of these advanced features is not supported in production. In production environments, all administrator assignments are restricted by licensing. NetIQ collects monitoring data in the audit database to ensure that production environments comply. Furthermore, NetIQ recommends that only one user be given the permissions of the Security Administrator.

The User Application Administrator is not a system role. For more information, see "User Application Administrator" on page 30.

Changing the Default Administrator Assignments after Installation

The following administrative accounts are assigned during the initialization of the User Application:

- Compliance Administrator
- Provisioning Administrator
- RBPM Configuration Administrator
- Resource Administrator
- Roles Administrator
- Security Administrator

Modifying the mappings for these administrative accounts in the configupdate utility after the installation and initialization process will not work in this release. The check for assigning the administrative roles happens only once. At this time, a property is set that keeps track of when these roles were assigned.

NOTE: To modify the default administrator assignments for the User Application, you must first edit the `configupdate.sh` or `configupdate.bat` file and change the `-edit_admin` property to `true`. You can then use configupdate to modify the default assignments.

If you want to modify the default assignments for the administrative roles without deleting the Driver (which would cause all role assignments to be removed), you need to perform one of the following actions:

- “Granting or Removing Assignments in the User Application” on page 65
- “Changing the Assignments in Configupdate Utility” on page 65
- “Changing the Default Administrator Assignments without an Administrator Account” on page 66

Granting or Removing Assignments in the User Application

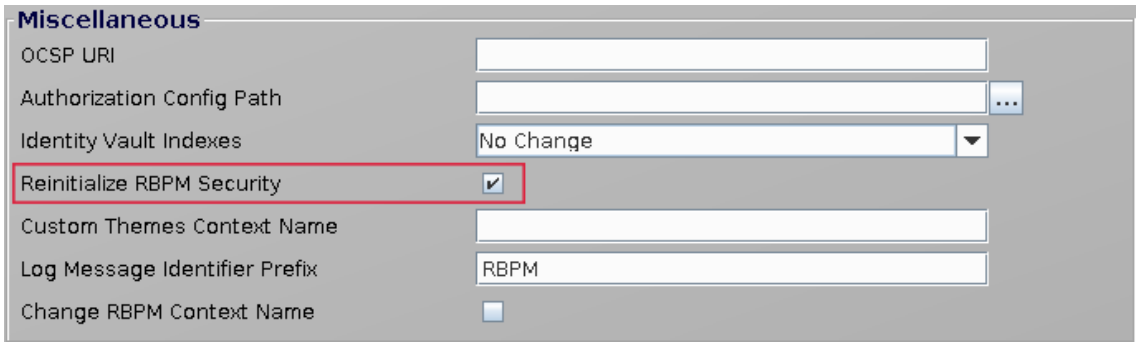
To grant or remove the role assignment through the User Application:

1. Log in to the User Application as the Security Administrator.
2. Go to the **Roles Catalog** on the **Roles and Resources** tab.
3. Select the administrative role you want to change (for example, the Provisioning Administrator).
4. Select **Edit**.
5. Select the **Assignments** tab.
6. If you want to remove the current assigned user, then select the user and press the **Remove** link.
7. To add a user, press the assign button where you will need to provide a description and the user to assign the role to and the press the **Assign** button.

Changing the Assignments in Configupdate Utility

To change any or all of the administrative assignments using Configupdate utility:

1. Stop the Application Server that the User Application WAR is deployed on.
2. Stop the User Application driver.
3. Stop the Role and Resource Service Driver.
4. Launch the configupdate utility.
5. Change the mappings for the administrative roles outlined above as required.
6. Click **Show Advanced Options**.
7. In **Miscellaneous**, check **Reinitialize RBPM Security** and click **OK**.



8 (Conditional) To remove the existing (default) users that have been granted the role assignment. Log in to iManager and remove the user from the role, then the role from the user.

9 Restart the User Application.

10 Restart the User Application driver.

11 Restart the Role and Resource Service Driver.

12 Access the User Application and in the logs you will see the administrative roles will be issued.

Changing the Default Administrator Assignments without an Administrator Account

The default administrator assignment settings are established at the time you initialize the User Application driver. After the driver has been initialized, you can change the default settings on the Administrator Assignments page, as long as your "admin" user account still exists. If the account has been deleted, deactivated, or moved to a different location, you will not be able to log in to make the new assignments. In this case, you need to reset the values in the configupdate utility or delete the initialization property in the User Application driver.

To change the administrator assignment values in the configupdate utility. See, "Changing the Assignments in Configupdate Utility" on page 65.

Alternatively, you can delete the initialization parameter in the User Application driver using iManager:

1 Log in to iManager.

2 In **Objects** tab, browse to ***Driver Set*** **> User Application Driver > AppConfig > AppDefs** and select **Configuration**.

3 In **General** tab, open **XMLData**.

4 Find and remove the `</property>` tag that contains the following `</key>` tag.

```
<key>com.novell.idm.security.domain-admin.initialized</key>
```

For example:

```
<property>
        <key>com.novell.idm.security.domain-admin.initialized</key>
        <value>20090831124642Z</value>
</property>
```

5 Click **OK**.

6 Restart the User Application driver and the Role and Resource Service driver.

7 Restart the Identity Applications.

8 Setting Up Logging in the Identity Applications

Logging is the main tool you use for debugging the identity applications configuration. The logging service provides facilities for writing, viewing, filtering, and listening for log messages. The Tomcat application server instances and subsystems, and applications that run on Tomcat or in client JVMs generate these log messages.

This sections discusses the following topics:

- "How Logging Services Help" on page 67
- "What Can Be Logged" on page 75
- "How Logging Works" on page 76
- "Types of Log Files" on page 78
- "Understanding the Log Format" on page 81
- "Configuring Logging" on page 83
- "Configuring Logging in a Cluster" on page 87
- "Logging to a Sentinel Server" on page 88
- "Using Log Files for Troubleshooting" on page 90
- "Log Events" on page 90

How Logging Services Help

A Tomcat server instance uses logging services to communicate its status and respond to specific events, including server startup and shutdown information, failures of one or more subsystems, errors, warning messages, access information on HTTP requests, and additional information. For example, you can use Tomcat's logging services to report error conditions or listen for log messages from a specific subsystem.

All administrative and end-user actions and events are logged to the server console and to Tomcat server's log file. This allows easy access to this information for security and operational purposes. Additionally, the audit log system provides the ability to monitor ongoing activities such as authentication activity, up time of the system, and so on. File logging is enabled by default.

The identity applications features are implemented in a layered architecture. Each feature uses one or more packages. Each package handles a specific area of a feature and has its own independent log level that obtains event messages from different parts of the application. The logs contain information about processing and interactions among identity applications components that occur while satisfying users and administrative requests and during general system processing. By enabling the correct log levels for various packages, an administrator can monitor how identity applications processes users and administrative requests. The package names are based on log4j conventions. The event messages include these package names indicating the context of the message output. The

logs include tags and values that allow the administrator to identify and correlate which package log entries pertain to a given transaction and user. Table 8-1 describes some of the features and the packages they use.

Table 8-1 *Identity Manager User Application Packages*

Feature	Description	Packages	Notes
Roles	Roles are permanently stored in the Identity Vault. For fast access to roles information, Identity Manager stores roles in a local cache called permission index. When a role is requested, the User Application queries the permission index for that role. When a role is modified through the User Application driver, the change is reflected in the permission index. For more information about roles, see "Understanding Roles" on page 24.	◆ com.novell.idm.nrf.service ◆ com.novell.idm.nrf.persist ◆ com.novell.srvprv.impl.vdata.model ◆ com.netiq.idm.rest.catalog	For troubleshooting any issues when a role is assigned, revoked, or expired, monitor the Roles and Resource driver log. `com.novell.srvprv.impl.vdata.model` is a verbose package when set to `Debug` log level. It generates messages for each object class and attributes present in Virtual Data Access (DAL). For example, it shows all DAL lookups. This can result in a large amount of logs. To limit the number of messages, you can set the log level to `Warn`. For more information about the messages generated by `com.novell.srvprv.impl.vdata.model`, see "Virtual Data Access Logging" on page 604. For troubleshooting issues related to managing roles, see "When a Role Is Requested" on page 609.

Feature	Description	Packages	Notes
Resources	Resources are permanently stored in the Identity Vault. For fast access to resources information, Identity Manager stores resources in a local cache called permission index. When a resource is requested, the User Application queries the permission index for that resource. When a resource is modified, the change is reflected in the permission index is updated. For more information about resources, see "Understanding Resources" on page 25.	• com.novell.idm.nrf.service • com.novell.idm.nrf.persist • com.novell.srvprv.impl.vdata.model	For troubleshooting any issues when a resource is assigned or revoked, monitor the Roles and Resource driver log.
Code Map Refresh	Code map is a local cache used by the User Application to store entitlements values for all connected systems from the Identity Vault. The User Application queries the Identity Vault for the drivers that are in running state and have entitlements. The User Application updates the User Application database at configurable intervals with entitlement changes. For more information about code map refresh, see "Configuring Default Resource Settings" on page 190.	• com.novell.idm.nrf.service • com.novell.idm.nrf.persist • com.novell.srvprv.impl.vdata.model	For troubleshooting any connected system issue, enable DSTrace on the driver. For viewing sample log messages related to code map refresh, see "When a Code Map Refresh Is Triggered" on page 607.
Proxy	Enables you to manage proxy configuration. Identity Manager stores proxy definition in the ProxyDefs container in the User Application driver. For more information about configuring proxy, see Acting on Behalf of Someone Else in *NetIQ Identity Manager - User's Guide to the Identity Applications*.	• com.novell.srvprv.impl.security.service • com.netiq.idm.rest.access • com.novell.soa.af.impl.persist • com.novell.srvprv.apwa.actions	When a user is designated as a proxy, check the audit events for any suspicious activity.

Feature	Description	Packages	Notes
Delegation	Enables you to manage delegation configuration based on a user`s availability. A delegate is another user that you can temporarily grant permission to view and resolve your workflow work items. A delegate can view his delegator tasks in the task page and act on them. Identity Manager stores delegate definitions in the DelegateeDefs container in the User Application driver. For more information about configuring delegation, see Chapter 16, “Creating and Managing Delegations,” on page 173.	◆ com.novell.srvprv.impl.security.service ◆ com.novell.srvprv.apwa.actions	When a user is made a delegate for another user, check the audit events for any suspicious activity that can occur through delegation.
Email-based Approvals	E-mail notifications inform Identity Manager users of tasks and events in the system. For example, Identity Manager can send an e-mail to approvers when an event or task requires an approval. For more information, see “Understanding Email-based Approval” on page 25.	com.novell.soa.notification.impl	For troubleshooting e-mail approval issues, see “Troubleshooting E-Mail Based Approval Issues” on page 620. For viewing sample log messages related to E-Mail notifications, see “Virtual Data Access Logging” on page 604.

Feature	Description	Packages	Notes
Database connectivity/ updates	Any schema changes made in the User Application are updated in the database when the User Application server is started and `com.netiq.idm.create-db-on-startup` flag is set to true in the `ism-configuration` properties file. When this flag is set, the database compares the existing schema with target schema and then updates the database schema. To update the database with any application configuration changes, you must set `com.netiq.idm.rbpm.updateConfig-On-StartUp` flag to true in the `ism-configuration` properties file.	com.novell.soa.persist	
Manage Featured Items (Landing page)	Allows you to manage application items on the landing page. You can quickly navigate to internal and external pages of the application. For more information, see Exploring the Dashboard in the *NetIQ Identity Manager - User's Guide to the Identity Applications*.	com.netiq.idm.icfg	
Client Settings	Allows you to manage client settings to control the behavior of the application. You can also modify the access rights and branding of the application for different set of users. For more information, see "Changing Identity Applications Client Settings" on page 131.	◆ com.netiq.idm.rest.access ◆ com.netiq.idm.settings	

Feature	Description	Packages	Notes
Workflow Tasks	A task can be controlled by a workflow process. A workflow process can include one or more steps that must be performed before Identity Manager can complete a task that is under workflow control. A job is a runtime instance of a workflow process. The Workflow Engine is responsible for managing and executing steps in a workflow and for keeping track of state information which is persisted in a database. For more information, see Part IV, "Configuring and Managing Provisioning Workflows," on page 215.	◆ com.novell.soa.af.impl.core ◆ com.novell.soa.af.impl.activity ◆ com.netiq.idm.rest.access	
Separation of Duties	Allows you to prevent users from being assigned to conflicting roles unless someone in your organization makes an exception for the conflict. To eliminate conflicts in role assignments, you perform certain management tasks such modify role definition and set up a proper approval process. For more information, see Chapter 17, "Separation of Duties Constraints," on page 175.	◆ com.novell.idm.nrf.service ◆ com.novell.idm.nrf.persist ◆ com.novell.srvprv.impl.vdata.model	
My Permission	A user can view a list of role and resource permissions assigned to him or for other users. For more information, see Viewing Your Permissions in the *NetIQ Identity Manager - User's Guide to the Identity Applications*.	◆ com.netiq.idm.rest.access ◆ com.netiq.idm.rest.access.util	
History	A user can review the status and history of the permission requests (role, resource, PRD) for himself or for other users. For more information, see Viewing Requests in the *NetIQ Identity Manager - User's Guide to the Identity Applications*.	◆ com.netiq.idm.rest.access ◆ com.novell.idm.nrf.persist ◆ com.netiq.idm.rest.access.util	

Feature	Description	Packages	Notes
Teams	You can perform team management tasks such as create, modify, and delete a team based on access privileges. Identity Manager stores team configuration in the TeamDefs container in the User Application driver. For more information about configuring Teams, see Managing Users in the *NetIQ Identity Manager - User's Guide to the Identity Applications*.	◆ com.netiq.idm.rest.access ◆ com.novell.idm.security.authorization.ldap ◆ com.novell.srvprv.spi.vdata.model ◆ com.netiq.idm.rest.access.util ◆ com.novell.idm.security.authorization.service	
Group	Allows you to manage groups. For example, you can create, modify and delete a group based on access privileges. For more information, see Managing Users, Groups, and Teams in the *NetIQ Identity Manager - User's Guide to the Identity Applications*.	◆ com.netiq.idm.rest.catalog ◆ com.novell.srvprv.spi.vdata.model	
Organization Chart	The Organization Chart page shows the hierarchy of users in your organization. A user can view the organization chart and quick information about the users based on the access rights set by the administrator. For more information, see Viewing an Organization Chart in the *NetIQ Identity Manager - User's Guide to the Identity Applications*.	◆ com.netiq.idm.rest.access ◆ com.novell.srvprv.impl.portlet.orgchart ◆ com.novell.srvprv.impl.servlet.service ◆ com.novell.soa.portlet ◆ com.netiq.idm.rest.access.util	
User Catalog	You can create, modify, and delete users. A new user is created under the base container configured for the user. Based on the access control list rights, the user information can be edited. The user attributes can be configured to view, edit, and search by using the client settings. For more information, see Managing Users in the *NetIQ Identity Manager - User's Guide to the Identity Applications*.	◆ com.netiq.idm.rest.access ◆ com.netiq.idm.infosrv ◆ com.novell.srvprv.impl.vdata.model ◆ com.netiq.idm.settings ◆ com.novell.idm.nrf.service ◆ com.novell.idm.security.authorization.service ◆ com.novell.idm.nrf.persist ◆ com.novell.idm.security.authorization.ldap ◆ com.netiq.idm.rest.access.util	

Feature	Description	Packages	Notes
Make a request	A user can request a permission for himself or for another user. The Request page directly fetches the permission from the Permission index. The requested permission is directly assigned or through an approval process. For more information, see Requesting Permissions in the *NetIQ Identity Manager - User's Guide to the Identity Applications*.	◆ com.netiq.idm.rest.access ◆ com.netiq.idm.infosrv ◆ com.novell.srvprv.spi.vdata.model ◆ com.novell.idm.nrf.ajaxservice ◆ com.novell.idm.nrf.service ◆ com.novell.idm.nrf.persist ◆ com.novell.idm.security.authorization ◆ com.novell.soa.af.impl.core ◆ com.novell.soa.af ◆ com.novell.idm.nrf.assignment	
Permission Index	Roles, resources, and PRDs are permanently stored in the Identity Vault. For fast access, Identity Manager stores this information on the User Application server in a set of cache files called Permission Index. When you install the identity applications, the process creates a permission index for the application server hosting the identity applications. When a request is issued, the identity applications query the permission index for the requested information.	◆ com.netiq.idm.cis ◆ com.netiq.cis.permindex ◆ com.netiq.idm.cis.permfilter ◆ com.sssw.fw.core ◆ com.netiq.uaconfig	Only applicable to NetIQ Identity Manager Dashboard and the new Dashboard.
Directory Abstraction Layer	The directory abstraction layer provides a virtual access to the Identity Vault data. You define a set of entities and their related attributes (virtual data) based on the Identity Vault objects that you want users to view, modify, or delete in the User Application. For more information, see Using the Directory Search in the User Application in *NetIQ Identity Manager - User's Guide to the Identity Applications*.	◆ com.novell.srvprv.impl.vdata.model	For viewing sample log messages related to Virtual Data Access, see "Virtual Data Access Logging" on page 604.

Feature	Description	Packages	Notes
Configuration	The **Configuration** tab on the Identity Manager Dashboard allows you to change the Identity Applications settings. Using this tab you can set the logging, caching, and clustering settings. You can also use this tab to assign administrators in Identity Applications. For more information, see Chapter 19, "Configuring Identity Applications Default Settings," on page 189.	• com.netiq.idm.rest.admin • com.novell.soa.af com.novell.soa.af.impl.core com.netiq.idm.settings.display com.netiq.logging com.sssw.fw.cachemgr.api • com.novell.idm.nrf.api	

The logs generated by the packages are primarily intended for debugging the software, although they can be used to detect any other software that is not behaving properly. System administrators and support personnel can identify and isolate problems caused by configuration errors, invalid user data, or network problems such as broken connections. However, component file logging is typically the first step in identifying software bugs.

Package logging is more verbose than audit logging. It increases the processing load. On a day-to-day basis, you are recommended to enable only log levels of error conditions and system warnings. If a specific problem occurs, logging can be set to **Info** or **Debug** to gather extra information needed to isolate and resolve the detected problem. When the problem is resolved, logging should be reconfigured to log only error conditions and system warnings.

What Can Be Logged

The identity applications functionality that deals with workflows (PRDs) and actions such as granting and managing of roles, resources, and entitlements can be logged. The log level for these features is controlled by configuring logging for the packages used by them. You can change log levels from the Logging page. For more information, see .

The identity applications functionality that executes on a client and does not directly execute in context of the User Application web server cannot be logged in the same way. For example, most of the form processing that occurs in the client. The action scripts defined on the form control's onLoad, onChange, or custom events execute in the browser of the user's client computer and not in the User Application web server. Therefore, if an error occurs while rendering the form or processing an action script, it cannot be directly logged. However, Identity Vault queries issued from a form or Start Activity can be logged for troubleshooting the identity applications features. To view the identity applications client errors and informational messages, click the Console tab or the Network tab under Developer Tools section in the client browser. It contains HTTP response codes for both success and failed requests. The identity applications allow you to record the outcome of a user's request and response.

Identity Applications allow you to log what happens with a user's request and response during certain times:

- Between the browser and the application server

- Between the application server and the User Application database
- Between the application server and the Identity Vault

You can configure the log files to include entries for the following events:

- Configuration
- Events processed by the identity applications components, such as authentication, role assignment, and resource access
- Error conditions

The log files help you determine which of the following reasons is responsible for a request failure:

- The browser did not send the required information
- Directory Access Layer or the Identity Vault did not send the web client browser the required information

To view Identity Manager processing events in Identity Manager drivers, use Trace. Specify appropriate trace values to the driver set and the drivers in Designer or iManager. For more information, see Viewing Identity Manager Processes in the NetIQ Identity Manager Driver Administration Guide.

How Logging Works

The following sections describe the identity applications logging environment and provide an overview of the logging process.

Terminology

Log4j has three main components: loggers, appenders, and layouts. These components work together to accomplish the following tasks:

- Record messages based on message type and level.
- Control how log messages are formatted and where they are reported at runtime.

Logger: In Log4j terminology, a logger is a named entity. Log4j defines a Logger class. A Logger object records messages for a specific subsystem or application component. An application can create multiple loggers, each with a unique name. In a typical usage of Log4j, an application creates a Logger instance for each application class that will emit log messages. Logger names are case-sensitive and they follow the Java package dot notation naming convention.

All loggers specific to the identity applications are defined in the `idmuserapp_logging.xml` file. You can set the severity level for each logger at any level in the hierarchy from the Logging Administration page or by editing the log4j file. For more information, see "Specifying the Severity Level for Commons Logging API Loggers" on page 85.

Appender: In Log4j terminology, an output destination is called an appender. Log4j defines appenders to represent destinations for logging output. You can define multiple appenders. For example, an application might define an appender that sends log messages to standard out, and another appender that writes log messages to a file. Additionally, you can configure individual loggers to write to zero or more appenders. For example, you can configure the loggers to send all logging messages (all levels) to a log file, but only Error level messages to standard out. To change

the destination of the log files, stop the identity applications and then change the settings in the Logging Location and Appender section of the `log4j.properties` file. Identity Applications provide a full suite of appenders offered by Log4j. For more information about appenders, see Log4j documentation.

The Console and File appenders are defined in the `tomcat-log4j.xml` file. The NAudit appender that is specific to the identity applications is defined in the `idmuserapp_logging.xml` file.

Layout: Log4j defines layouts to control the format of log messages. Each layout specifies a particular message format. A specific layout is associated with each appender. This lets you specify a different log message format for standard out than for file output if required.

Components for Logging

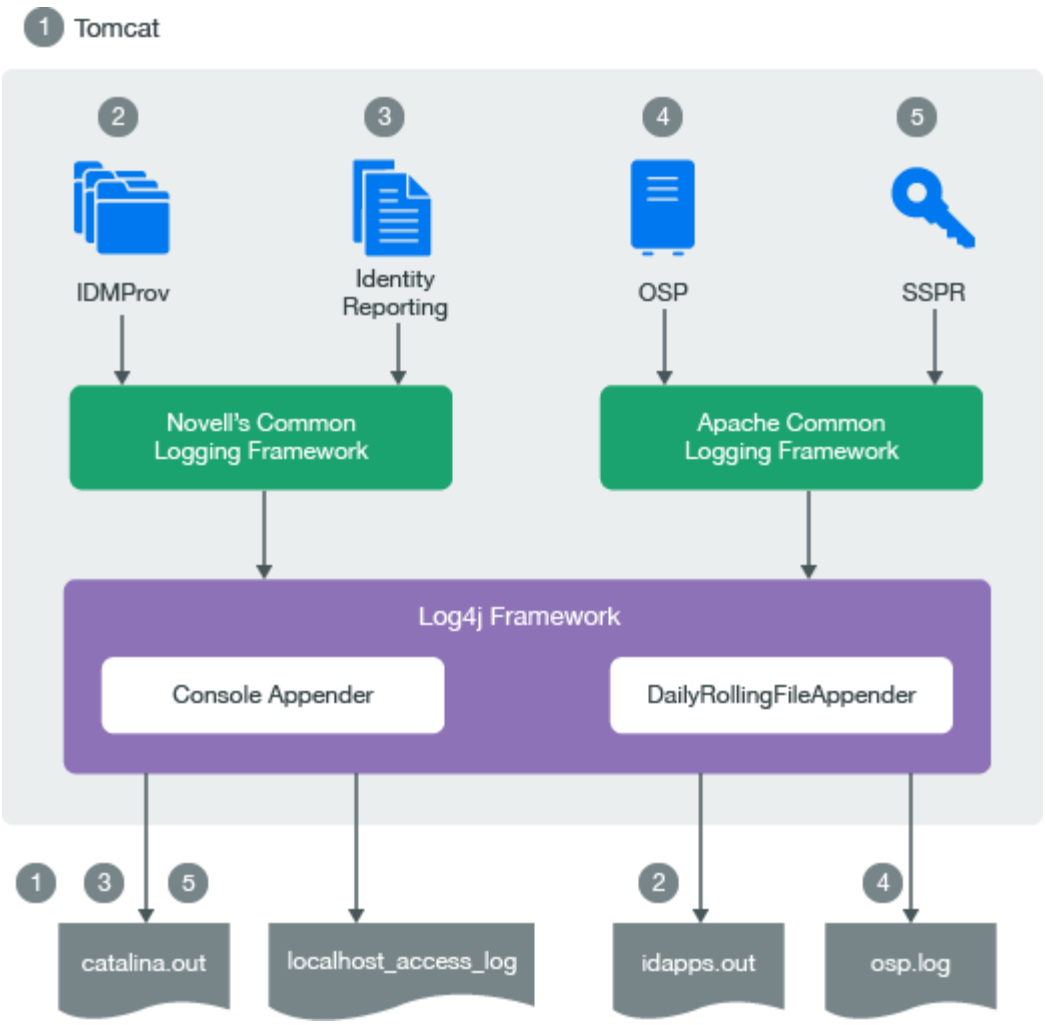
A logging system includes the following basic components:

- A component that produces log messages
- A component that distributes (publish) log messages

The Tomcat subsystems use log4j to produce messages. By default, Tomcat supports Java based logging to distribute messages. The LoggingHelper class provides access to the java.util.logging.Logger object used for server logging. The Java Logging APIs can be used to add custom handlers, filters, and formatters. Alternatively, you can configure Tomcat to use Log4j APIs to distribute log messages.

How Logging Works

The identity applications support logging by using a custom-developed logging framework that integrates with log4j, an open-source logging package distributed by The Apache Software Foundation. In identity applications environment, Tomcat subsystems and identity applications components send log requests to the Logger objects. The Logger objects then assign LogRecord objects, which are passed to Appender objects for publication. By default, the logger objects log messages to the system console and to the Tomcat server's log file at `Info` logging level and above. Events are logged to all activated loggers.



- The Tomcat server's log messages are directed to `catalina.out and idapps.out`.
- User Application's log messages are directed to `idapps.out`.
- Identity Reporting's log messages are directed to `catalina.out`.
- OSP's log messages are directed to `osp.out`.
- SSPR's log messages are directed to `catalina.out`.

Custom appenders like NAuditAppender are created to handle log messages in order to convert the messages to a specific format and send them to the configured auditing service. To configure event message output to an auditing service, see "Configuring Identity Applications Default Settings" on page 189.

To configure logging, see "Configuring Logging" on page 83.

Types of Log Files

An identity applications framework uses several components such as OSP, SSPR, Dashboard, User Application driver, and Role and Resource Service driver. Each component has its own logging configuration that defines the default log levels and appender configuration for that component. Logs for all identity applications components including OSP and Identity Reporting are logged to the

catalina.out file while the Localhost log records the interactions between the Tomcat server and the client. Some components such as OSP maintain additional log files that help in auditing their individual interactions. This section discusses different log files that are generated in an identity applications environment and what each of them contains.

Difference Among Catalina, Application, and Localhost Log Files

Catalina Log: This is the global log. It records information about events such as the startup and shutdown of the Tomcat application server, the deployment of new applications, or the failure of one or more subsystems. The messages include information about the time and date of the event and the ID of the user who initiated the event.

The catalina.log file contains all log messages that are written to Tomcat's system.out and system.err streams. Tomcat's internal log statements use the java.util.logging package (juli) to log. The default destination for that log is standard.out.

The catalina.out file can include:

- Uncaught exceptions printed by java.lang.ThreadGroup.uncaughtException(..)
- Thread dumps, if you requested them via a system signal

Each Tomcat server instance prints a subset of its messages to standard.out or idapps.out file. The idapps.out log file is specific to User Application. The userapp-log4j.xml file stores the logging configuration for User Application, which directs all log messages to idapps.out.

The catalina.out file is located on the computer that hosts the Tomcat server instance. Each server instance has its own catalina.out file. By default, the catalina.out file is located in the logs directory under Tomcat's root directory. For example, /opt/netiq/idm/apps/tomcat/logs/catalina.out. To view messages in the catalina.out file, log in to the computer hosting Tomcat and use a standard text editor.

NetIQ recommends that you do not modify the log files by manually editing them. Modifying a file changes the timestamp and can confuse log file rotation. In addition, editing a file might lock it and prevent it from recording information from the Tomcat server.

Some operating systems enable you to redirect standard out to some other location. By default, a server instance prints only messages of Info severity level or higher to standard out. You can modify the severity threshold as a logging configuration so that the server prints more or fewer messages to respective log files.

Localhost Log : This is the log for all HTTP transactions between the client and the application server. The log file is named as, localhost_access_log.<date of log generation>.txt file. The default location and rotation policy for this log is the same as catalina.out file.

Application Log: Each identity application component is responsible for its own logging. Tomcat provides no support for application logs. Each component will have its own logging configuration where default log levels and appender configurations are defined. These logging configuration files are placed under \conf directory of Tomcat server.

Additional Log Files

The `catalina.out` log messages and log files communicate events and conditions that affect Tomcat server's operations. Logs for all identity applications components including OSP and Identity Reporting are also logged to the `catalina.out` file. This file also records the interactions between the Tomcat server and the client.

Some subsystems and components also maintain additional log files that help in auditing their individual interactions. The following list describes some of the additional log files:

- Logging information from the User Application is also logged into the `idapps.out` file. However, this file does not contain Tomcat server specific information.
- OSP logs are additionally stored in a separate file, `osp-idm-<date of log generation>.log` file located in `/opt/netiq/idm/apps/tomcat/logs/` directory. Logging is turned off by default and must be enabled in the `setenv.sh` file in the `/TOMCAT_INSTALLED_HOME/bin/` directory.
- Identity Reporting logs are additionally stored in `/var/opt/netiq/idm/log/`.
- SSPR logs are stored in `/opt/netiq/idm/apps/sspr/sspr_data/logs/SSPR.log`.
- The HTTP subsystem keeps a log of all HTTP transactions between the client and the application server in a text file, `localhost_access_log.<date of log generation>.txt` file. The default location and rotation policy for this log is the same as the catalina.out file. You can set the attributes that define the behavior of HTTP access logs for your server.
- By default, logs for User Application driver and Role and Resource Service driver are added to DSTrace. The trace is turned off by default and must be enabled in the driver configuration by using Designer or iManager or console. When enabled, DSTrace displays messages related to operations that the driver performed or tried to perform, at the level of detail specified by the driver trace level, as the engine processes the events. The driver trace level affects only the driver or driver set where it is set. You can also specify to write the trace information for a driver to a separate file. For more information, see Viewing Identity Manager Processes in the *NetIQ Identity Manager Driver Administration Guide*.
- Each server has a transaction log which stores information about committed transactions coordinated by the server that may not have been completed. Tomcat uses the transaction log when recovering from a system crash or a network failure. You cannot directly view the transaction log. The file is in a binary format.
- The auditing service records information from a number of security requests, which are determined internally by the security framework. The service also records the event data associated with these security requests and the outcome of the requests. Each server writes auditing data to its own log file in the server directory.
- The JDBC subsystem records various events related to JDBC connections, including registering JDBC drivers and SQL exceptions. The events related to JDBC are written to the server log, such as when connections are created or refreshed or when configuration changes are made to the JDBC objects.

- The JMS logging is enabled by default when you create a JMS server. The identity applications use Apache ActiveMQ as a JMS provider. Logs can be found in the ActiveMQ installed path at data/activemq.log. The identity applications rely on a Java Message Service (JMS) persistent store to persist e-mail messages. If JMS is not properly configured, any e-mail messages in the memory queue will be lost if the application server is shut down.
- The Hibernate framework writes log messages in different categories and log levels. For example, it logs messages for establishing connections with the database, executed SQL statements, or cache interactions. Hibernate logging is enabled by default at Info level. The events related to JDBC subsystem are written to the server log. All Hibernate related information is logged when the data is persisted in the application.

Understanding the Log Format

The identity applications have a specific format for file log entries. To improve log entry readability, the log entries in the `catalina.out` files use standard elements. This facilitates the use of non-interactive stream-oriented editors such as sgrep, sed, awk, and grep. The first part of each message begins with locale-formatted timestamp followed by a data portion that contains information specific to the log entry. The data portion is the most flexible part of a log entry.

A log entry has the following fields:

```
time-date-stamp [Severity] [Subsystem] [Message Text]
```

The following entry is an example entry that is logged when a user has requested for a role:

```
2017-03-08 08:43:10,660 INFO
com.novell.idm.nrf.service.RoleManagerService- [RBPM] [Role_Request]
Requested by cn=David.Scully,ou=Active,ou=People,o=acme, Target DN:
CN=Kaitlin.Demore,OU=active,OU=People,O=acme, Source
DN:cn=PennDOT_Vehicle_Certification,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplicatio
n,cn=idm46,ou=services,o=acme, Request DN:cn=20160308084310-
15da49b28ddf4ee1b7d71b4ce220c080-
0,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm46,ou=se
rvices,o=acme, Request Category: 10, Request Status: 0, Original Request
Status: 0, Correlation ID: UserApp#UserStartWorkflow#082080ec-5b10-498f-
b87d-28825ab63278
```

Most log entries do not use the optional line breaks ([\n]). Notice that the time-date-stamp, the severity, subsystem, and message text are on the same line so that stream-oriented editors that use only one line (such as grep) can be used to locate related log entries.

If a message is logged within the context of a transaction, the message text contains the Correlation ID assigned to the transaction.

The Tomcat server uses the host computer's default character encoding for the messages it writes.

Message Fields

A Tomcat server message contains a consistent set of fields as described in the following table. In addition, if your applications use Tomcat logging services to generate messages, those messages will contain these fields.

Table 8-2 *Fields in a Log Entry*

Field	Description
Time-date-stamp	Time and date when the message originated in a format that is specific to the locale. The JVM that runs a Tomcat server instance refers to the host computer operating system for information about the local time zone and format. The date and time is specified in the W3C profile format of ISO 8061. It has the following fields: year-month-day-T-hour-minutes-seconds-time zone. The Z value for the time zone indicates that the time is specified in UTC.
Severity	Indicates the degree of impact of the event reported by the message such as warning, informational, or debug. In the example log entry, the level of severity is Info.
Subsystem	Indicates Tomcat's subsystem or the type of the module that was the source of the message. For example, RBPM or Java Messaging Service (JMS). In the example log entry, this field contains the following string: `com.novell.idm.nrf.service.RoleManagerService- [RBPM] [Role_Request]`

Message text	A description of the event or condition specific to the log entry. It can be as simple as an informational string, such as the string in the example log entry:
	
	`Requested by cn=David.Scully,ou=Active,ou=People,o=acme, Target DN: CN=Kaitlin.Demore,OU=active,OU=People,O=acme, Source DN:cn=PennDOT_Vehicle_Certification,cn=Application Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm46,ou=services,o=acme, Request DN:cn=20160308084310-15da49b28ddf4ee1b7d71b4ce220c080-0,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm46,ou=services,o=acme, Request Category: 10, Request Status: 0, Original Request Status: 0,`
	
	If a message is logged within the context of a transaction, the message text contains an identifier assigned to the identity applications transaction. Identity applications transactions are actions such as authenticating a user, processing a request for a role, and request for access to a resource.
	
	An identifier is assigned to each identity applications transaction.
	
	If a user requests access to multiple resources, multiple request objects are created and each request is given a separate Correlation ID. Each request is processed separately and status of each can be seen in the user request history.
	
	The example log entry contains the following Correlation ID:
	
	`UserApp#UserStartWorkflow#082080ec-5b10-498f-b87d-28825ab63278`

Message Severity

The severity attribute of a log message indicates the potential impact of the event or condition that the message reports. The following table lists the severity levels of log messages from the identity applications subsystems, starting from the lowest to the highest level of impact.

Level	Description
Fatal	The least detail. Writes fatal errors to the log.
Error	Writes errors (plus all of the above) to the log.
Warn	A suspicious operation or configuration has occurred but it might not affect normal operation. Writes warnings (plus all of the above) to the log.
Info	Used for reporting normal operations; a low-level informational message. Writes informational messages (plus all of the above) to the log.
Debug	Writes debugging information (plus all of the above) to the log.
Trace	Used for most detailed messages. You can configure this level to report the request path of a method. Writes tracing information (plus all of the above) to the log.

You can set a log severity level on loggers and appenders. When set on a logger, none of the appenders receives any events that are rejected by that logger. For example, if you set the log level to `Info` on a logger, none of the appenders will receive `Warn` level events. When you set a log level on the appender, the restriction only applies to that appender and not to other appenders. For example, if you turn `Error` off for the File appender, no `Error` messages are written to the log file. However, `Error` messages are written to the `standard.out` file.

```
<logger name="com.sssw" level="INFO" additivity="true">
            <appender-ref ref="NAUDIT"/>
        </logger>
<logger name="com.netiq" level="DEBUG" additivity="true">
            <appender-ref ref="NAUDIT"/>
        </logger>
```

You set log levels for loggers using the Logging Administration page or the log4j file. For more information, see “Configuring Logging Settings in Identity Manager Dashboard” on page 85. Loggers can also be configured through APIs. You can only enable or disable an appender.

The identity applications modules generate many messages of lower severity and fewer messages of higher severity. For example, under normal circumstances, they generate many `Info` or `Trace` messages. If your application uses Tomcat logging services, it can use an additional severity level of `Debug`.

Configuring Logging

You can configure logging to troubleshoot errors or to receive notification for specific events. For example, configure logging to perform the following activities:

- Stop recording of Debug and Info messages in the log file.
- Allow recording of Info level messages from the HTTP subsystem in the log file.
- Configure an appender to publish messages only whose severity level is Warning or higher.
- Track log information for individual servers in a cluster.

You may need to configure the logging.properties file in some cases.

The following sections describe basic configuration tasks:

- "Understanding Logging Configuration" on page 84
- "Understanding the Log Level Settings" on page 84
- "Specifying the Severity Level for Commons Logging API Loggers" on page 85
- "Configuring Logging Settings in Identity Manager Dashboard" on page 85
- "Editing the log4j Files" on page 86
- "Managing Log File Size" on page 87

Understanding Logging Configuration

When you enable logging, a logging request is sent to subscribed appenders. Tomcat provides appenders for sending log messages to the `standard.out` file and the server log (`catalina.out`) file. You can control logging for each type of handler by filtering log messages based on severity level and other criteria. For example, the Stdout Handler has a Notice threshold severity level by default. Therefore, Info and Debug level messages are not sent to the `standard.out` file.

By default, event messages are logged to the system console and to the Tomcat server's log file at `Info` logging level and above. Events are logged to all activated loggers.

The default behavior of the Tomcat server is to limit the console log4j appender to display log messages with a verbosity of Info or less. To see log messages for more verbose levels (for example, Debug), you need to examine the server log file. Notice that the low threshold settings, such as Debug are extremely verbose and will increase Tomcat's startup time.

The following sections discuss different ways of configuring the logging behavior.

Understanding the Log Level Settings

Console logging involves synchronized writes. Therefore, logging can become a processor usage issue and concurrency impedance. You can change the priority value default setting to Error, on a Tomcat server, by modifying the setting in the `<installdir>/Tomcat/server/IDMProv/conf/tomcat-log4j.xml` file. Locate the root node that looks similar to this:

```
<root>

   <priority value="INFO"/>

   <appender-ref ref="CONSOLE"/>
   <appender-ref ref="FILE"/>
</root>
```

Change the priority value to:

```
<root>
      <priority value="ERROR"/>

      <appender-ref ref="CONSOLE"/>
      <appender-ref ref="FILE"/>
</root>
```

Assigning a value to the root ensures that any appenders do not have a level assigned inherit the root's level.

Specifying the Severity Level for Commons Logging API Loggers

If you are using the Commons Logging API, logger names follow the Java package dot notation naming convention. For example, a logger name can be `com.acme.Barlogger`, corresponding to the name of the classes in which it is used. Each dot-separated identifier appears as a node in the Logger tree. In this case, the logger named com.acme is a parent of the logger named `com.acme.Barlogger`.

You can configure the severity for a package or for any logger at any level in the tree. For example, if you specify the severity level for package `com.acme=Warn`, then `Fatal` and `Error` messages from the child nodes of this package will be blocked. You can override the severity level of a parent node by explicitly setting a value for a child node. For example, if you specify the severity level for `com.acme.Barlogger=Debug`, all log messages from `Barlogger` will be allowed, while `Fatal` and `Error` messages will be filtered for other child nodes under `com.acme`.

You can specify the severity level for a package or a logger in the following ways:

- In Identity Manager Dashboard, go to **Configuration > Logging** and change the log level settings. The changes will be immediately applied. When you restart Identity Applications, the changes are not preserved. To persist the changes for subsequent sessions, select **Persist the logging changes**. Alternatively, modify the `log4j.properties` file.
- Edit the `log4j.properties` file. Your changes will take effect when you restart the User Application. Identity Applications preserves this configuration for subsequent sessions.

If you change, enable, or disable logging, you need not restart the identity applications to apply the changes.

Configuring Logging Settings in Identity Manager Dashboard

The **Logging** page shows a list of all currently defined loggers. On this page, you can:

- Add a new logger for a class or package name. For more information, click ⓘ in Identity Manager Dashboard.
- Remove a logger for a class or package name. For more information, click ⓘ in Identity Manager Dashboard.
- Set the logging level (Fatal, Error, Warn, Info, Debug, Trace) for each class or package name.
- Reset all logging levels.

All logging configuration cannot be changed in the **Logging** page, such as Tomcat server specific configuration and appender configuration. For making such changes, stop the identity applications and then edit the `log4j.properties` file.

You can change the log level of the packages individually by searching a package name. If you want to change the log level for all the packages:

1. Select **Change log level for the listed packages**.
2. Select the log level from the list.

Table 8-3 *Types of Log Levels*

Level	Description
Fatal	The least detail. Writes fatal errors to the log.
Error	Writes errors that can cause system processing to not proceed.
Warn	Logs potential failures, but the impact on execution is minimal. Warnings indicate that you should be aware that this event is happening and might want to make a configuration change to avoid it.
Info	Logs informational messages. No execution or data impact occurred.
Debug	Includes debugging information.
Trace	The most detail. Writes tracing information (plus all of the above) to the log.

NOTE: By default, the log level is set to **Info** for all the packages.

3 (Conditional) To retain these changes after restarting the application server, select **Persist the logging changes**.

4 Click **Apply**.

NOTE: The portal functionality and export or import of portal content within the User Application is discontinued from this release. If you have the packages corresponding to these features, manually remove the packages from the `idmuserapp_logging.xml` file.

Editing the log4j Files

The configuration settings for the identity applications logging are stored in the `idmuserapp_logging.xml` in the install directory on the Tomcat server. The log4j configuration settings are contained in `tomcat-log4j.xml` in the install directory. To configure the logging levels and other settings permanently, stop the Tomcat server and change the settings in these files.

NOTE: The Console and File appenders are defined in `tomcat-log4j.xml`. The NAudit appender that is specific to the identity applications is defined in `idmuserapp_logging.xml`. All loggers specific to the identity applications are defined in `idmuserapp_logging.xml`.

To change the log level in the `tomcat-log4j.xml` file, open the file in a text editor and locate the following entry at the end of the file:

```
<root>
    <priority value="INFO" />
      <appender-ref ref="CONSOLE" />
      <appender-ref ref="FILE" />
  </root>
```

Assigning a value to root ensures that any log appenders that do not have a level explicitly assigned inherit the root level (in this case, Info). For example, the File appender does not have a default threshold level assigned. It assumes the root's threshold level.

The possible log levels used by log4j are Debug, Info, Warn, Error, and Fatal, as defined in the `org.apache.log4j.Level` class. Inattention to the proper use of these settings can be costly in terms of performance.

A good rule of thumb is to use Info or Debug only when debugging a particular problem.

Any appender included in the root that does have a level threshold set, should set that threshold to Error, Warn, or Fatal unless you are debugging something.

The performance hit with high log levels has less to do with verbosity of messages than with the simple fact that console and file logging, in log4j, involve synchronous writes. An AsyncAppender class is available, but its use does not guarantee better performance. These are known issues of Apache log4j.

The default log level of Info in the log configuration file for the identity applications is suitable for many environments. However, for a performance intensive environment, you can change the entry as follows:

```
<root>
      <priority value="ERROR"/>
      <appender-ref ref="FILE"/>
</root>
```

For a fully tested and debugged production setup, enabling Info and Console logging are not needed. For more information about log4j, see Apache Logging Services.

Managing Log File Size

By default, event messages are logged to both of the following:

- The system console of the application server where the identity applications components are deployed
- A log file on that Tomcat server. For example: `/opt/netiq/idm/apps/tomcat/logs/catalina.out`

  This is a rolling log file. By default, the server rotates the file based on a time interval of 24 hours. However, you can instruct the server to rotate the file over to another file after it reaches a certain size by specifying the size in the `log4j.appender.R.MaxFileSize` property in the `$TOMCAT_HOME/lib/log4j.properties` file. It does not rotate the local server log file when you start the server.

To cause the immediate rotation of the log file, change the appender configuration in the `log4j.properties` file.

By default, the rotated files are stored in the same directory where the log file is stored. You can specify a different directory location for the archived log files in the `log4j.properties` file.

Configuring Logging in a Cluster

This section includes tips for configuring logging in a Tomcat cluster.

- "Tomcat Logging" on page 88
- "User Application Logging" on page 88

Tomcat Logging

You can configure Tomcat for logging in a cluster. To enable logging for clusters, you need to edit the `tomcat-log4j.xml` configuration file, located in the `\conf` directory for the Tomcat server configuration (for example, `\server\IDM\conf`), and uncomment the following section at the end of the file:

```
<!--  Clustering logging
  -->
- <!--
 Uncomment the following to redirect the org.jgroups and
      org.tomcat.ha categories to a cluster.log file.
   <appender name="CLUSTER"
class="org.tomcat.logging.appender.RollingFileAppender">
     <errorHandler class="org.tomcat.logging.util.OnlyOnceErrorHandler"/>
     <param name="File" value="${tomcat.server.home.dir}/log cluster.log"/>
     <param name="Append" value="false"/>
     <param name="MaxFileSize" value="500KB"/>
     <param name="MaxBackupIndex" value="1"/>
     <layout class="org.apache.log4j.PatternLayout">
       <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>
     </layout>
   </appender>
   <category name="org.jgroups">
     <priority value="DEBUG" />
     <appender-ref ref="CLUSTER"/>
   </category>
   <category name="org.tomcat.ha">
     <priority value="DEBUG" />
     <appender-ref ref="CLUSTER"/>
   </category>
  -->
```

You can find the `cluster.log` file in the `log` directory for the Tomcat server configuration (for example, `\server\IDM\log`).

User Application Logging

The logging configuration is not propagated to all servers in cluster. For example, if you use the Logging administration page on a server in a cluster to set the logging level for `com.netiq.afw.portal.aggregation` to Trace, this setting is not propagated to the other servers in the cluster. You must individually configure the level of logging messages for each server in the cluster.

For more information, see “Configuring Logging Settings” on page 192.

Logging to a Sentinel Server

To enable NetIQ Sentinel logging in your Identity Manager environment, you need to configure the Platform Agent on your application server and then enable Sentinel logging.

Configuring the Platform Agent

The Platform Agent is required on any client that reports events to Sentinel. You configure the Platform Agent through the `logevent` configuration file. This file provides the configuration information that the Platform Agent needs to communicate with the Sentinel server. The default location for this file, on the application server, is:

- Linux: `/etc/logevent.conf`
- Windows: `/<WindowsDir>/logevent.cfg` (Usually `c:\windows`)

To configure the Platform Agent:

1. Configure the Platform Agent on your application server.

   Specify the following properties:

   Loghost: The IP address or DNS name of your Sentinel server. For example:

   ```
   LogHost=xxx.xxx.xxx.xxx
   ```

   LogJavaClassPath: The location of the lcache jar file `NauditPA.jar`. For example:

   ```
   LogJavaClassPath=/opt/netiq/idm/NAuditPA.jar
   ```

   LogCacheDir: Specifies where lcache stores cache files. For example:

   ```
   LogCacheDir=/opt/netiq/idm/naudit/cache
   ```

   LogCachePort: Specifies on which port lcache listens for connections. The default is 288, but in a Linux server, set the port number greater than 1000. For example:

   ```
   LogCachePort=1233
   ```

   BigData Specifies the maximum number of bytes that the client will allow. Larger amounts of logging data will be truncated. The default value is 3072 bytes, but you should change this to at least 8192 bytes to handle a typical form that has approximately 15 fields on a half page.

   ```
   LogMaxBigData=8192
   ```

   IMPORTANT: If your data is very large, you might want to increase this value. If you are logging events that include digital signatures, it is critical that the value of LogMaxBigData be large enough to handle the data being logged.

   Specify any other settings needed for your environment.

   NOTE: You must restart the Platform Agent any time you change the configuration.

   ---

2. Restart the Platform Agent for the changes to take effect.

Enabling Sentinel Logging

1. Log in to Identity Manager Dashboard as a User Application Administrator.
2. Go to **Configuration > Logging**.
3. Select **Enable naudit service**.

4 (Conditional) To allow log events in CEF format, select **Enable CEF format** and specify the sentinel server details.

5 (Conditional) To save the changes for subsequent restart of the Tomcat server, make sure **Persist the logging changes** is selected.

Using Log Files for Troubleshooting

The following example has trace messages logged to the `catalina.out` file when a code map refresh cycle is triggered.

```
2017-08-29 16:05:05,500 [INFO] CodeMapEngine [RBPM] Refreshing the
Entitlement CODE MAP tables...
2017-08-29 16:05:05,499 [TRACE] BestLocaleServletFilter [RBPM] Using
Resource-Group[common-resgrp] with bestLocale[en] for /IDMProv/
GwtServiceRouter
2017-08-29 16:05:05,510 [DEBUG] DirXMLDriverDAO [RBPM] Entering
getDriversEnabledForMappings() mappingType=2
2017-08-29 16:05:05,600 [DEBUG] DirXMLDriverDAO [RBPM] Exiting
getDriversEnabledForMappings()
2017-08-29 16:05:05,600 [INFO] CodeMapEngine [RBPM] Done refreshing the
Entitlement CODE MAP tables
```

The descriptions of the log messages are sequentially listed below:

- The first message indicates that a code map refresh is in process. This is the first entry when a code map refresh is triggered.
- The second message specifies the entitlement values are persisted to the best suited locale.
- The third message provides information about the drivers enabled with entitlements.
- The fourth message indicates that the process is exiting after obtaining the information about the drivers enabled with entitlements.
- The fifth message indicates that the code map is updated with the entitlements values from the drivers and the process is completed.

Log Events

The identity applications log a set of events automatically from workflow, search, detail, and password requests. By default, the following events are automatically logged to all active logging channels:

Table 8-4 *Logged Events*

Event ID	Process	XDAS Event	Severity
31400	Detail portlet		Info
31401			Info
31410	Change Password portlet		Error
31411			Info

Event ID	Process	XDAS Event	Severity
31420	Forgot Password portlet		Error
31421			Info
31430	Search portlet		Info
31431			Info
31440	Create portlet		Info
31520	Workflow		Error
31521			Info
31522			Info
31523			Info
31524			Info
31525			Info
31526			Info
31527			Info
31528			Info
31529			Info
31534			Info
31535			Info
31537			Info
3152A			Info
3152B			Info
3152C			Info
31533			Info
31538			Info
31539			Info
3153A			Info
3153B			Info
3153C		XDAS_AE_CREATE_DATA_ITEM	info
3153D		XDAS_AE_CREATE_ROLE	Info

Event ID	Process	XDAS Event	Severity
3152D	Provisioning		Error
3152E			Info
3152F			Info
31530			Error
31531			Info
31532			Info
31550		XDAS_AE_CREATE_SESSION	Info
31551		XDAS_AE_CREATE_SESSION	Info
31450	Security Context		Info
31451			Error
31452			Info
31453			Error
31454			Info
31455			Error
31456			Info
31457			Error
31458			Info
31459			Error
3145A			Info
3145B			Error
3145C			Info
3145D			Error
3145E			Info
3145F			Error
31600	Role Provisioning	XDAS_AE_APPROVAL_REQUESTED	Info
31601		XDAS_AE_APPROVAL_REQUESTED	Error
31610	Role Assignment Request		Info
31611			Error
31612			Info

Event ID	Process	XDAS Event	Severity
31613		XDAS_AE_CREATE_DATA_ITEM_ASSOC	Info
31614		XDAS_AE_TERMINATE_PEER_ASSOC	Info
31615		XDAS_AE_TERMINATE_PEER_ASSOC	Error
31620	User Entitlement	XDAS_AE_CREATE_DATA_ITEM_ASSOC	Info
31621		XDAS_AE_CREATE_DATA_ITEM_ASSOC	Error
31622		XDAS_AE_TERMINATE_DATA_ITEM_ASSOC	Info
31623		XDAS_AE_TERMINATE_DATA_ITEM_ASSOC	Error
31624			Error
31630	Role Management		Info
31631			Error
31632			Info
31633			Error
31634			Info
31635			Error
31640			Info
31641			Error
31642			Info
31643			Error
31644			Info
31645			Error
31646		XDAS_AE_MODIFY_DATA_ITEM_ATT	Info
31647		XDAS_AE_MODIFY_DATA_ITEM_ATT	Error

```
00031665,Resource Provisioning
00031666,Resource Provisioning Failure

00031600,Role Provisioning
00031601,Role Provisioning Failure

00031677,Create Resource Association Failure
00031678,Delete Resource Association
00031679,Delete Resource Association Failure
0003167A,Modify Resource Association
0003167B,Modify Resource Association Failure

#^GROUP^Engine events logged from vrdim^00030001-00030032
00030001,Status Success
00030002,Status Retry
00030003,Status Warning
00030004,Status Error,Channel
00030005,Status Fatal
00030006,Status Other
00030007,Search
00030008,Add Entry
00030009,Delete Entry,Channel
0003000A,Modify Entry
0003000B,Rename Entry
0003000C,Move Entry
0003000D,Add Association
0003000E,Remove Association
0003000F,Query Schema
00030010,Check Password
00030011,Check Object Password
00030012,Change Password
00030013,Sync,Channel
00030014,Input XML Document
00030015,Input Transformation Document
00030016,Output Transformation Document
00030017,Event Transformation Document
00030018,Placement Rule Transformation Document
00030019,Create Rule Transformation Document
0003001A,Input Mapping Rule Transformation Document
0003001B,Output Mapping Rule Transformation Document
0003001C,Matching Rule Transformation Document
0003001D,Command Transformation Document
0003001E,Publisher Filter Transformation Document
0003001F,User Agent Request
00030020,Resync Driver
00030021,Migrate
00030022,Driver Start
00030023,Driver Stop
00030024,Password Sync
00030025,Password Reset
00030026,DirXML Error
00030027,DirXML Warning
00030028,Custom Operation
00030029,Clear Attribute
0003002A,Add Value - Modify Entry
```

```
0003002B,Remove Value
0003002C,Merge Entries
0003002D,Get Named Password
0003002E,Reset Attributes
0003002F,Add Value - Add Entry

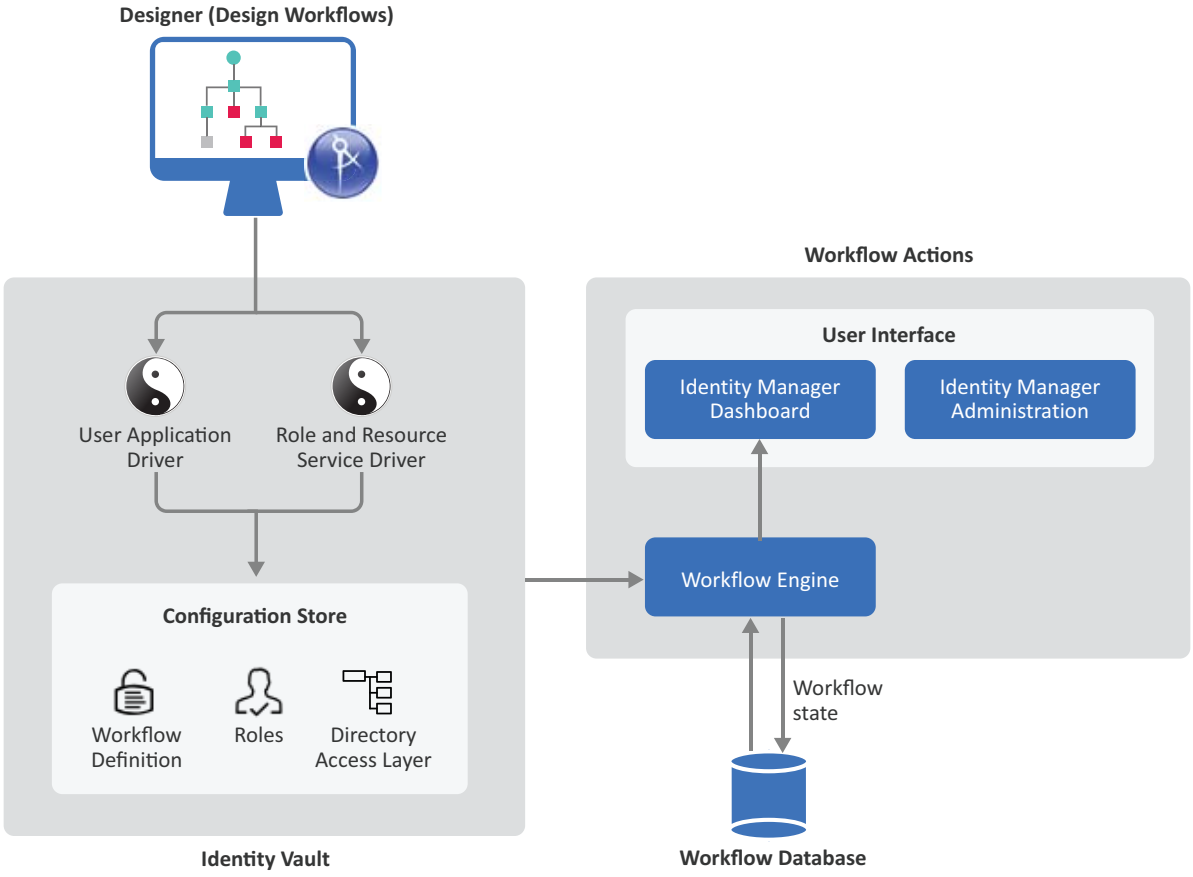
#^GROUP^Job events logged from vrdim^000303E4-000303E7
000303E4,Job Result Aborted
000303E5,Job Result Error
000303E6,Job Result Warning
000303E7,Job Result Success

#
#^GROUP^Server events Logged from DXevent^000307D0-000307E2
000307D0,Config:Log Events
000307D1,Config:Driver Cache Limit
000307D2,Config:Driver Set
000307D3,Config:Driver Start Option
000307D4,Driver Resync
000307D5,Migrate Application
000307D6,Shim Password Set
000307D7,Keyed Password Set
000307D8,Remote Loader Password Set
000307D9,Regenerate Key Pair
000307DA,Get Server Certificate
000307DB,Cache Utility
000307DC,Check Object Password
000307DD,Initialize Driver Object
000307DE,Notify Job Update
000307DF,Open Driver Action
000307E0,Queue Driver Event
000307E1,Start Job
000307E2,Abort Job

#^GROUP^Remote Loader^00030BB8-00030BBB
00030BB8,Remote Loader Start
00030BB9,Remote Loader Stop
00030BBA,Remote Loader Connection Established
00030BBB,Remote Loader Connection Dropped
```

9 Tuning the Performance of the Applications

Identity Applications rely on diverse technologies with many interactions.



This section discusses common aspects that can enable you to optimally tune the performance of provisioning components and Tomcat application server. You can use this information as a reference for starting your performance tuning. For a good understanding of potential areas where tuning can improve performance, monitor your application usage patterns, loads, and hardware specifications, and then track specific performance issues.

Several tools are available on the Internet to monitor Java applications. Standard Java JDK comes with two graphical user interface-based monitoring tools: JConsole and VisualVM. These tools are free and easy to install. VisualVM provides advanced monitoring features than JConsole. VisualVM enables to analyze the thread execution and profile CPU and memory usage of the JVM requests. For more information about these tools, see JDK Tools (https://docs.oracle.com/javase/8/docs/technotes/tools/). The following are a few Linux-based examples of monitoring local applications that are running on the same system as VisualVM and remote applications that are running on other systems:

Monitoring a local application

Use the process ID (PID) to monitor applications that are running locally that VisualVM can connect to.

Monitoring a remote application

You can monitor CPU usage, heap usage, threads, memory, and classes on applications running on other systems.

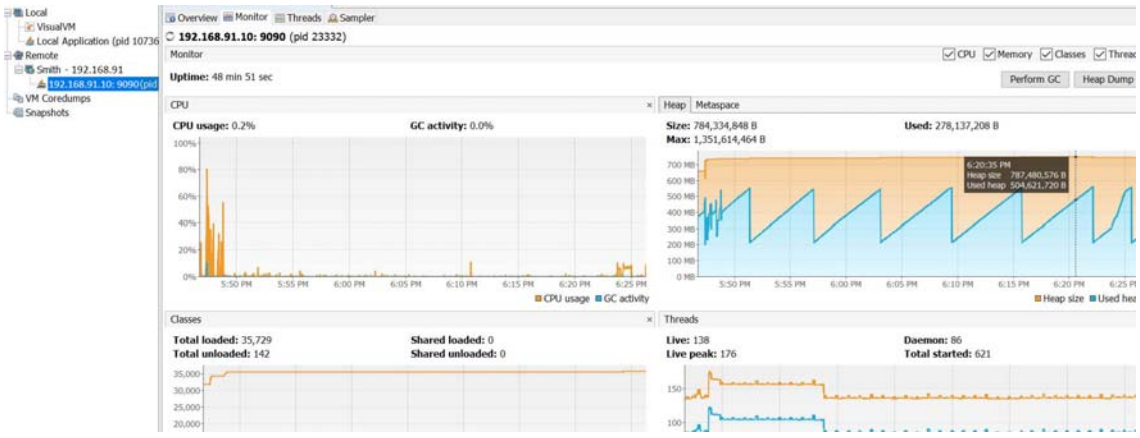
Perform the following actions to set up a connection with the remote system:

1. Add the following system properties to the `CATALINA_OPTS` entry of the `setenv.sh` file in the `/tomcat/bin/` directory:

   ```
   -Dcom.sun.management.jmxremote
   -Dcom.sun.management.jmxremote.port=<Monitoring_Port>
   -Dcom.sun.management.jmxremote.authenticate=false
   -Dcom.sun.management.jmxremote.ssl=false
   ```

2. Launch JVisualVM from `<JAVA_HOME>/bin/jvisualvm`.
3. Add the remote host by using the port configured and add the JMX connection.

JVisualVM is composed of four tabs: Overview, Monitor, Threads, and Sampler. The below figure shows the Memory tab that provides information about memory consumption and memory pools.



NOTE: The focus of the section is tuning and not troubleshooting. Effective troubleshooting involves identifying the clues and root cause of the problem, and then making corrections. You must first attempt to troubleshoot the problem before thinking about tuning. For more information about troubleshooting, Chapter 41, “Troubleshooting,” on page 603.

Increasing the Heap Size

If your application takes a longer time to respond, you may need to optimize the memory reserved for the heap. For example, when Identity Applications are deployed in a large environment with approximately a million permissions, it may take a few minutes (1-2) to load the permission index. If the delay is not because of the hardware used, you can use the following Java virtual machine options to tune the heap:

- If you run out of heap memory (not due to a memory leak), increase `-Xmx`.
- If you run out of native memory, you may need to decrease `-Xmx`.
- To tune the size of the heap for the young generation for JVM, modify `-Xmn`. For more information, see the Oracle Java (https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/sizing.html) documentation.

The heap size value is determined by the amount of memory available in the computer. Initial heap size is 1/64th of the computer's physical memory or reasonable minimum based on platform (whichever is larger) by default. The initial heap size can be overridden using `-Xms`. Maximum heap size is 1/4th of the computer's physical memory or 1 GB (whichever is smaller) by default. The maximum heap size can be overridden using `-Xmx`.

If you installed Identity Applications on a computer with a minimum 4 GB memory, and you want to improve the performance of the system under heavy load, set `Xmx` to `2048m` for both minimum and maximum heap size in the `/opt/netiq/idm/apps/tomcat/bin/setenv.sh` or `C:\NetIQ\idm\apps\tomcat\bin\setenv.bat` file. For example, `-Xms2048m -Xmx2048m`. For minimum memory requirements, see the Identity Manager 4.7 System Requirements (https://www.netiq.com/documentation/identity-manager-47/system-requirements-identity-manager-47x/data/system-requirements-identity-manager-47x.html) page. For general guidance about how heap size is calculated, see the Oracle Java (https://docs.oracle.com/javase/10/gctuning/JSGCT.pdf) documentation.

The amount of heap memory allocated to the Java virtual machine can impact performance. For example, if the `Xmx` value is too low (is set lower than the amount of live data in the JVM), it will force frequent garbage collections in order to free up the space (RAM). You may experience excessive page file swapping if `Xms` or `Xmx` value is more than the physical memory of the computer.

Increasing the Stack Size for Recursive Workflows

If you have workflows that are recursive in nature (that execute loops), you might see a StackOverflow error at the execution time.

For example, if you have a Provisioning Request Definition with a branch activity, multiple condition activities, and a merge activity, Identity Applications may become unresponsive due to the following reasons:

- Exceeded stack size limit

  Java does not effectively handle the stack space for recursive type functions. The default value for the stack size in the JVM is `1024K`. Therefore, you must increase the stack size for the JVM to `4M` depending on the recursive logic defined in the workflow. To increase the stack size, define the new value in the `-Xss` setting in the JAVA_OPTS entry in your Tomcat start script file (`/`

opt/netiq/idm/apps/tomcat/bin/setenv.sh or C:\NetIQ\idm\apps\tomcat\bin\setenv.bat). For example, to set the stack size to 4M, change the setting to -Xss4M.

```
JAVA_OPTS="-server -Xss4M -Xms512M -Xmx512M"
```

- Exhausted database connection

  An exhausted database connection can lead to a delayed response from the application when the application is performing database intensive operations such as executing complex workflows and bulk task approvals. In such cases, you must analyze the thread dump to detect if the database communication is degrading your JVM performance. If the delay is caused by the database communication, set the value for the maxTotal property for your database in the server.xml file located at /opt/netiq/idm/apps/tomcat/conf or C:\NetIQ\idm\apps\tomcat\conf to 200. For example:

```
<Resource auth="Container" driverClassName="org.postgresql.Driver"
factory="com.netiq.tomcat.jdbc.pool.CustomBasicDataSourceFactory"
initialSize="10" maxTotal="200" maxIdle="10" minIdle="10"
name="shared/IDMUADataSource" password="<passsword>"
testOnBorrow="true" type="javax.sql.DataSource" url="jdbc:postgresql://
<ip-address>:port/idmuserappdb" username="idmadmin"
validationInterval="120000" validationQuery="SELECT 1"/>
```

Ensuring Concurrent Access from Multiple Clients

In a medium to large environment, you might have 50 or more clients accessing the server concurrently. To prevent operational failures, configure the following settings:

- "Decreasing the Session Time-out" on page 100
- "Increasing the Number of Maximum Open Files" on page 101
- "Increasing the Number of User Processes" on page 102
- "Adjusting the Threadpool Size" on page 102
- "Increasing the Database Connection Pool" on page 103

Decreasing the Session Time-out

The **session time out** represents the amount of time users can leave a page unattended in their web browser before the server displays a session-time-out warning. You must tune this value to match the server and usage environment in which the application runs. In general, the session time out should be as short as practicable. If a 5 minute time out does not impact your business, this time period would allow the server to release memory twice as early as it would if the time-out value were 10 minutes. This improves the performance and scalability of the web application.

You can define time out for web application, LDAP, and Tomcat sessions. In Identity Applications, only LDAP and web application session time outs are used.

- The web application session time out value is represented by the com.netiq.idm.session-timeout property in the ism-configuration.properties file. The default value is 1200 milliseconds.

- The LDAP session time out value is represented by the `com.novell.ldap.timeout` property in the `ism-configuration.properties` file. The default value is 600000 milliseconds.
- The Tomcat session time out represents the amount of time Tomcat retains the information pertaining to a user's session, when the user does not perform any action on the web page. All Tomcat servers provide a default `web.xml` file in the `tomcat_home/conf/` directory. To configure the session time out globally for the entire web server, locate the `<session-config>` configuration setting and change the value of the `session-timeout` tag. The Tomcat session time out value is overwritten by the web application session time out value for every user session.

The default time out values for a web application, Tomcat, and LDAP sessions are different. They need not be same. For example, a user needs more time on the Identity Applications user interface than a query that is issued to retrieve information from the Identity Vault.

Consider the following points when adjusting the session time out:

- Long session time-outs can cause the Tomcat server to run out of memory if many users log in for a short period. This is valid for application server that has too many open sessions.

  For example, if the session time out value is 10 minutes and approximately 10 users log in per minute and those users are idle and do not log out during this time, you are likely to have 100 concurrent users at any point in time. If a single user consumes 1 MB of heap, 100 MB of memory is used by those idle users. When the maximum time out value is reached, the idle users need to log in again to use the application. If a query is in progress and it hits the maximum time out value, the query is stopped. You must log in again for the query to continue.

  The recommended value is 3 to 5 minutes.
- When a user logs in to Identity Applications, an LDAP connection is created for the user and the user is bound to the session. Thus, the more sessions that are open, the greater the number of LDAP connections that are held. The longer the session time out, the longer these connections are held open. Too many open connections to the LDAP server (even if they are idle) can cause system performance degradation.

  In addition to a short time out interval, increase the number of open files on the Linux server. If you exceed the open file limit, check your `catalina.out` logs for errors. For more information, see “Increasing the Number of Maximum Open Files” on page 101.
- If the server starts experiencing out-of-memory errors, and the JVM heap and garbage collection tuning parameters have already been optimally tuned for the server and usage environments, consider lowering the session time out.

Increasing the Number of Maximum Open Files

Lowering the session time out quickly clears the LDAP connection threads, which improves performance.

If Tomcat reports the `Too many open files` exception when the number of concurrent logged-in users is large, consider increasing the limit of open files (`ulimit –n`) in the `server.xml` file depending on the concurrent sessions expected on the server. Ensure that the host operating system is able to handle the increase in the number of open files. For example, on a server with 100 concurrent connections, increase the number of open files to 10000.

Increasing the Number of User Processes

Increase the number of user processes (`ulimit -u`) for the user that runs the Tomcat process on all Tomcat servers. For example, if you have 30000 concurrent connections, setting the number of user processes to 128502 works fine.

Adjusting the Threadpool Size

The configured connectors determine how browser and other clients access Identity Applications deployed on Tomcat. Tomcat 8 provides a default HTTPS connection, which leads to better thread management with longer running requests. The default maximum thread pool size within Tomcat 8 is 200.

Keep in mind that many active threads at the same time can slow down Identity Applications, and even the server hosting Identity Applications. Therefore, you must set the maximum number of threads that can be created to meet your requirements by using the `maxThreads` property in the `server.xml` file. For example, specify `maxThreads="1000"` for an `https` connector. On a computer with four processors, you can set this value between 800 and 1000. For example, set it to 800 if you expect a limited number of users to access the application and 1000 when a large numbers of users are anticipated to access the application. If the value exceeds the number of threads required, the thread pool shrinks as the volume decreases.

Connectors are configured by using the `minProcessors`, `maxProcessors`, `acceptCount`, and `enableLookups` properties in the `server.xml` file. Below is an extract of connector configuration from the `server.xml` file in which some of these properties are assigned specific values.

```
<Connector port="8090" maxHttpHeaderSize="8192"
               maxThreads="400" minSpareThreads="150" maxSpareThreads="300"
                connectionTimeout="2"
  acceptCount="50"
  maxKeepAliveRequests="400"
  disableUploadTimeout="true" />
```

The `acceptCount` property defines the number of pending requests awaiting processing by the server. By default, it is `10`. You must change it when a high rate of incoming connection requests result in connection failures. For example, leave it at default if you expect a limited number of requests and change upto 50 to accommodate a large numbers of requests when several users access the application at the same time. However, setting it too high can overload Tomcat, which can cause problems for all requests.

Increase `maxKeepAliveRequests` if you have sufficient capacity. Using persistent connections improves performance and prevents wasting CPU resources in reestablishing HTTP connections.

If the system slows down with a high thread count, analyze the performance and thread usage under the anticipated usage load and then adjust the value. For example, to obtain the number of threads of a process, run `grep Threads /proc/<PROCESS_PID>/status`.

If the workload of a single server is high, you can cluster the Tomcat instance to allow the thread pool to be virtually distributed across the nodes of the cluster. Each server in the cluster will need less within its thread pool.

For complete documentation of the HTTP Connector configuration, see http://tomcat.apache.org.

Increasing the Database Connection Pool

Identity Applications communicate with the database through a JDBC connection pool. A connection pool starts with a small number of connections. As clients demand for more connections increases, there may not be enough connections in the pool to satisfy the requests. The Tomcat server creates additional connections and adds them to the pool until the maximum pool size is reached.

For example, when multiple users access the Tasks page at the same time, Identity Applications connect to the User Application database and query for the task list. Tomcat allocates a resource from the database connection pool to make the connection with the database. When all connections are relatively busy and the connection pool is empty, Identity Applications wait until a connection is released.

To configure a JNDI data source that uses connection pooling in Tomcat, add the following ResourceLink global element in the `conf/context.xml` file:

```
<ResourceLink global="shared/IDMUADataSource" name="jdbc/IDMUADataSource"
type="javax.sql.DataSource" />
```

In this example, the JNDI context is created with `jdbc/IDMUADataSource` name, which is a type of DataSource. The database configurations are defined in Tomcat's `server.xml` file in the `url`, `username`, `password`, and `driverClassName` attributes. The connection pooling properties are defined in `maxTotal`(`maxActive` before Tomcat 8), `maxIdle`, and `minIdle` attributes.

Review the following while configuring the JDBC connection pool:

- The number of concurrent tasks that will execute at the same time.
- The other runtime components. Each runtime component works in conjunction with the other runtime components. NetIQ recommends setting the size of the initial connection pool to `10`. To change it, modify the following entry in Tomcat's `server.xml` file:

  ```
  <Resource auth="Container" driverClassName="org.postgresql.Driver"
  factory="com.netiq.tomcat.jdbc.pool.CustomBasicDataSourceFactory"
  initialSize="10" maxActive="50" maxIdle="10" maxWait="30000"
  minIdle="10" name="shared/IDMUADataSource"
  password="srNOGbFupaKGrFfhULugQw==:KggiF61gvadcEpxqlCt5gQ==:yFZtb+h0bk
  5iQ7LUniIsRA==" testOnBorrow="true" type="javax.sql.DataSource"
  url="jdbc:postgresql://164.99.91.252:5432/idmuserappdb"
  username="idmadmin" validationInterval="120000" validationQuery="SELECT
  1" />
  ```

- The number of connections in the pool must allow each executing task to retrieve and update task and event data throughout the lifetime of the task.
- The database uses prepared statements. You must configure the prepared statement cache for the database that you are using to store the data.

View Request Status Search Limit

By default, the **View Request Status** action retrieves up to 10,000 request objects. If a user attempts to retrieve a larger result set, the user will see a message indicating that the limit has been reached. In this case, the user should narrow the search (by specifying a particular user or status, for example) to limit the number of objects returned in the result set. Note that when a user applies a filter to a role name, the filter limits what the user sees and its order, not the number of objects returned.

To change the maximum number of request objects retrieved:

1. Log in to iManager.
2. Navigate to the User Application driver.
3. Under **EntityDefs.DirectoryModel.AppConfig**, locate the `sys-nrf-request` object.
4. Modify the **XmlData** attribute for the object to the following value:

   ```
   <search-max>10000</search-max>
   ```

5. Save your changes.

Decreasing the LDAP Socket Cleanup Interval

Identity Applications tracks and clears all the LDAP sockets and references at the specified interval. By default, this interval is set to 60 minutes. A simple user search will not take time. When too many LDAP requests are handled, CPU utilization and memory usage of the system increases leading to performance issues. In this case, you might need to decrease the LDAP socket cleanup interval manually. A small interval cleans the memory regularly and decreases the memory footprint of the process. Otherwise, the socket objects remain in the memory and cause out of memory issues.

To decrease the LDAP socket cleanup interval:

1. Open the `ism-configuration.properties` file that is located at:

   Linux: `/opt/netiq/idm/apps/tomcat/conf/`

   Windows: `C:\NetIQ\idm\apps\tomcat\conf`
2. Set the `com.novell.idm.ldap.socket.cleanup.interval` property to `10` minutes.

   For example,

   ```
   com.novell.idm.ldap.socket.cleanup.interval=10
   ```

3. Restart the Identity Applications service.

   ```
   systemctl restart netiq-tomcat
   ```

Optimizing LDAP Connection with Identity Vault

Identity Applications use LDAP connections to communicate with the Identity Vault server. The LDAP time out value represents the maximum time after which an LDAP connection to the Identity Vault is timed out by the LDAP server. The default value is 600000 miliseconds (10 minutes). The connection is timed out as soon as the 600'th second is reached regardless of whether it is an idle connection or it is in the middle of processing a query. If an LDAP query is still running and has not completed when it reaches the 600'th second, the connection is closed between Identity Vault and Identity Applications. Therefore, if your LDAP query is expected to take more time, increase the value of the `com.novell.ldap.timeout` property in the `ism-configuration.properties` file.

You must change the LDAP connection time out value to match the Identity Vault usage in your environment depending on how much time out period can you afford. For example, if your query is not performing as expected or the data size that you are expect the query to return is large, increase the time out value. Decreased time out value allows the server to release unused resources relatively quickly, which improves the performance and scalability of Identity Applications.

Indexing Attributes in the Identity Vault

LDAP queries can be a bottleneck in a heavily utilized directory-server environment. To maintain a high level of performance with large numbers of objects, create indexes for the attributes that are frequently used in Identity Vault searches. An index is a set of keys arranged in a way that significantly speeds up the task of finding any particular key within the index. This allows to efficiently retrieve the objects whose attributes are indexed during searches without having to search the entire database every time a search is issued. When a complex query is run against objects with indexed attributes, the query returns relatively faster.

The Identity Vault ships with a set of indexes that provide basic query functionality. These default indexes are for the following attributes:

```
Aliased Object Name
cn
dc
Equivalent to Me
extensionInfo
Given Name
GUID
ldapAttributeList
ldapClassList
Member
NLS: Common Certificate
Obituary
Reference
Revision
Surname
uniqueID
uniqueID_SS
```

When you install Identity Manager, the default directory schema is extended with new object class types and new attributes pertaining to Identity Applications. The Identity Applications specific attributes are not indexed by default. For better performance, you might find it useful to index some of those attributes (and perhaps a few traditional LDAP attributes as well), particularly if your user container contains over 5,000 objects.

The general idea is to index only those attributes that you know are regularly queried, which could be different attributes in different production environments. If you already know which attributes you are likely to use in your searches, you must index those attributes. For example, if you know that users of your `org chart` are likely to perform searches based on the isManager attribute, you can try indexing that attribute to see if the performance is enhanced.

NOTE: As a best practice, it is recommended that you index, at a minimum, the manager and isManager attributes.

Indexes can be of type presence, value, and substring indexes.

- **Value** matches the entire value or the first part of the value of an attribute. For example, value matching could be used to find entries with a LastName that is equal to “Jensen” and entries with a LastName that begins with “Jen.”
- **Presence** requires only the presence of an attribute rather than specific attribute values. A query to find all entries with a Login Script attribute would use a presence index.

- **Substring** matches a subset of the attribute value string. For example, a query to find a LastName with “der” would return matches for Derington, Anderson, and Lauder. A substring index is the most resource-intensive index to create and maintain.

A compound index is a new type of index. You can think of it as a value index on more than one attributes. It stores the values of the attributes as part of the key for the index. For more information, see “Enabling Compound Index on Identity Vault Attributes” on page 106.

The type of index you create, depends upon if the search is looking for any value (*), or a specific value such as person or admin. For example, use a presence index when a * is used in the query. Use a value index when a specific value is used in the query. Although indexes improve search performance, additional indexes also add to directory update time. As a general rule, create new indexes only if you suspect performance issues are related to a particular directory lookup.

You can create, manage, and delete indexes in iManager (**eDirectory Maintenance > Indexes**). For more information, see “Index Manager” (https://www.netiq.com/documentation/edirectory-91/edir_admin/data/a5tuuu5.html) in the *eDirectory Administration Guide*.

For more information about eDirectory performance tuning, see “Maintaining eDirectory” (https://www.netiq.com/documentation/edirectory-91/edir_admin/data/a5zek7a.html) in the *eDirectory Administration Guide*.

Enabling Compound Index on Identity Vault Attributes

Compound index is a value index covering multiple attributes. Compound index was primarily added to support sorting on multiple attributes for the Server Side Sort control, you can use it for improving the performance of searches if the attributes being searched are part of a compound index.

When compound index is not enabled, you are likely to encounter issues where results are not displayed in the Identity Applications user interface. For example, the User Catalog page does not display the results.

The Identity Applications installation program automatically creates the following indexes:

```
0$GnSnIndex$0$0$0$1$Given Name$Surname
0$SnGnIndex$0$0$0$1$Surname$Given Name
0$CnSnIndex$0$0$0$1$CN$Surname
0$TitleSnIndex$0$0$0$1$Title$Surname
0$OuSnIndex$0$0$0$1$OU$Surname
0$LSnIndex$0$0$0$1$L$Surname
0$mailSnIndex$0$0$0$1$Internet EMail Address$Surname
0$telephoneNumberSnIndex$0$0$0$1$Telephone Number$Surname
```

To create and manage indexes and compound indexes, use eDirectory’s ndsindex utility. For compound indexes, specify multiple attributes separated by $ sign in the ndsindex utility.

For example, to create a value index with the name MyIndex on the email address and surname attributes, enter the following command:

```
ndsindex add –h myhost –D cn=admin, o=mycompany –w password –s cn=myhost,
o=netiq 'MyIndex;email address$surname;value'
```

Alternatively, you can create a compound index by using an LDIF file. For example, sort users on custom attributes.

1. Create an LDIF file with the compound index entries.

   1a `dn: CN=linux-32ep,OU=servers,O=system` (Your Server DN)

   ```
   changetype: modify

   add: indexdefinition

   indexdefinition: 0$gnsnindex$0$0$0$1$given name$surname (specify
   the correct order)
   ```

   1b `dn: CN=linux-32ep,OU=servers,O=system` (Your Server DN)

   ```
   changetype: modify

   add: indexdefinition

   indexdefinition: 0$sntitleindex$0$0$0$1$Title$surname
   ```

2. Run the following command to create indexes:

   ```
   ice -S LDIF -a -c -f comp.ldif -D LDAP -s 164.99.178.47 -p 389 -d
   cn=admin,ou=sa,o=system -w novell -F –B
   ```

3. (Optional) Check if the indexes are online:

   ```
   ndsindex list -D cn=admin,ou=sa,o=system -w novell -s CN=linux-
   32ep,OU=servers,O=system
   ```

Consider the following while creating compound indexes:

- Although you can specify multiple attributes for compound index, try not to add more than three attributes. As you go on adding the attributes, the performance of the searches is decreased. In case of value type compound index, you can add a maximum of five attributes.
- You are recommended to connect ndsindex utility to the same eDirectory server where the index has been added.

NOTE

- An index with ancestor id can only be created with value index type. Presence and Substring index types are not supported with ancestor id.
- Database size increases after creating index with ancestor id.

Comparison with Other Indexes

The cost of managing compound indexes in terms of time is the same as any other value index. Any modification such as addition or deletion of value requires the index to update.

The order of the number of attributes added to a compound index determine disk space used by the index.

The key size for a compound index is higher because all attribute values are added to the key. If other attributes are present, then a key for those attributes is added to the index. This can lead to bigger keys and increased number of keys as compared to normal value indexes.

NOTE: Having a high number of indexes has an adverse performance impact on modify operations because it requires updating the indexes with the modified attribute. This is not specific to compound indexes.)

Any modification (addition/deletion of value) would require the index to be updated.

For more details on how to create or manage compound indexes, see, Examples for Compound Indexes (https://www.netiq.com/documentation/edirectory-91/edir_admin/data/a6qjdjx.html#akiydau) in the *eDirectory Administration Guide*.

Sample Error Message

If an attribute is part of compound indexes, the following error message is displayed in the Manage Users page.

```
Sorting functionality does not work for //attribute key//attribute. Please
contact the system administrator for more details".
```

The following error message is displayed in the catalina.out with complete exception trace.

```
OperationNotSupportedException: [LDAP: error code 53 - Unwilling To
Perform].
```

Managing the eDirectory Database Cache Objects Retrieved from the Identity Vault Server

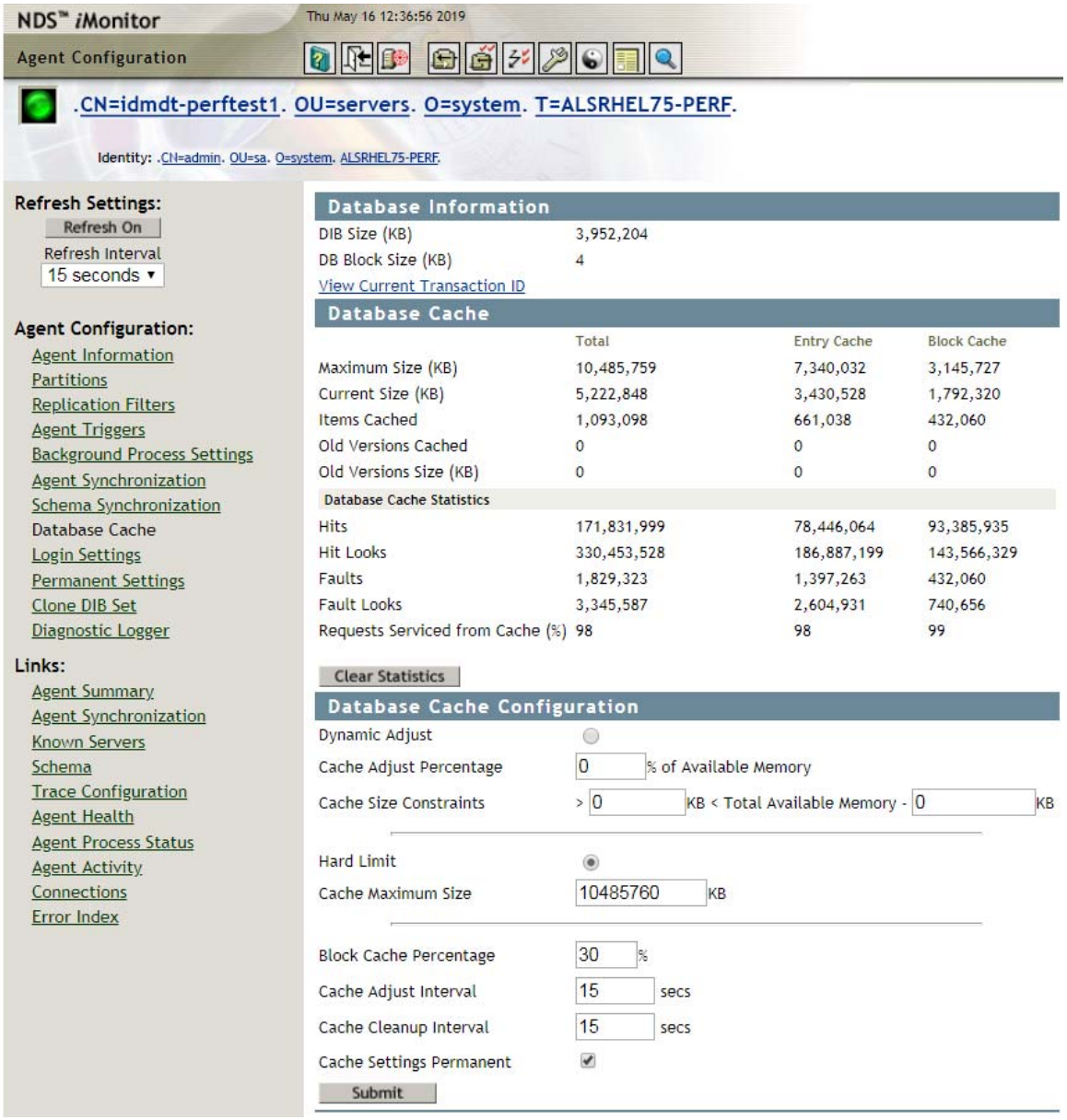
The Database Cache allows you to configure and monitor the cache segments that are used by the database subsystem of eDirectory. The easiest way to monitor, control, and configure Database Cache settings is through iMonitor.

iMonitor can run on all platforms supported by eDirectory. This utility lets you monitor your servers from any location on your network where a Web browser is available. You can monitor your eDirectory environments based on a partition, replica, or server. You can also examine all tasks, when they occur, their results, and how long they take. iMonitor provides several options for tuning the Database Cache settings.

To access the Database Cache page in iMonitor, enter the following in your address bar and log in as administrator with the fully distinguished name, or as an administrator equivalent:

```
http: //<server ip>:8008/nds/agent?config=CacheCtl
```

The information that displays on the Database Cache page helps you determine whether you have an adequate cache size and suggests how much cache to allocate. For most applications of the directory, the default cache size and settings are adequate.



The Database Information section of the page shows the size of your DIB in KBs. In this example, the DIB is about 4 GBs. It is hard-coded to use 8 GBs of RAM for the cache. The DIB has about 400,000 users and 100,000 groups.

To monitor the cache statistics, locate the Database Cache section on the page.

About 5.2 GB of the 10 GB allocated is actually used by the cache. When it reaches the maximum allocated size, eDirectory divides the memory into equal parts, as the Max Entry and Max Block lines show. To change this configuration, use the **Block Cache percentage** option at the bottom of the page.

You can assign nearly fifty percent of the memory to the cache in normal cases. A new installation of eDirectory on a Linux system defaults to 200 MBs. You can change it for your requirements.

You can view and change the cache settings from the Database Cache Configuration section. eDirectory supports static and dynamic memory allocation. Static allocation is predictable and does not pose maintenance overhead for the underlying operating system. You are recommended to use static memory allocation, but consider your environment before making the selection. For more information about controlling the cache memory consumption of eDirectory, see the *eDirectory Tuning Guide* (https://www.netiq.com/documentation/edirectory-91/edir_tuning/data/bqmivb8.html).

You can also view status of hits and misses to the cache from this section. In this example, 98% of the hits were served from the cache. Like all caches, it may take some time for the cache to build up after a service restart.

You can link to the Agent Summary, Agent Information, Agent Configuration, Trace Configuration, DSRepair, Reports, and Search pages from any iMonitor page by using the icons in the Navigator frame. You can also log in or link to the NetIQ Support Web page from any iMonitor page. For more information about iMonitor and the features it provides, see iMonitor Features (https://www.netiq.com/documentation/edirectory-91/edir_admin/data/b1gkpdzf.html#b1h7wmyw) in the *eDirectory Administration Guide*.

10 Customizing the Identity Applications for Your Enterprise

Identity Manager provides several tools for localizing or customizing the content in the identity applications user interface. This section helps you perform the following activities:

- "Linking the Dashboard to External Applications" on page 111
- "Customizing the Look of the User Interfaces" on page 112
- "Localizing the Text in the Interfaces" on page 113
- "Adding a Language to the Identity Applications" on page 119
- "Configuring User Names" on page 125
- "Configuring Email Notification Templates for the Dashboard" on page 127
- "Configuring Forgot Password? Functionality" on page 128
- "Ensuring that Characters Display Properly in Role Report PDF Files" on page 128
- "Ensuring that Dates Display Correctly in Norwegian" on page 129
- "Configuring Client Settings Mode" on page 130
- "Changing Identity Applications Client Settings" on page 131

For more information about...	See...
Setting the preferred locale	"Specifying Locales and Localization Resource Groups" in the *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*
Localizing email templates	"Adding Localized Email Templates" on page 253
Localizing challenge questions	"Security Best Practices" in the *NetIQ Identity Manager Security Guide*
Localizing provisioning objects or customizing their display text, such as: • Directory abstraction layer objects • Provisioning request definitions • Workflow activity display names	"Localizing Provisioning Objects," in the *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*

Linking the Dashboard to External Applications

You can modify the **Applications** page in the Dashboard to display all the applications, activities, and permissions that you want users to access. To ensure that users can access Identity Reporting and SSPR, add the following types of links to the **Applications** page:

Activity	Link to...
Run identity reports	Identity Reporting
Manage password	Self-service Password Reset

Managing Featured Items

You can modify the **Applications** page to display all the applications, activities, and permissions that you want users to access. By default, the identity applications provide a **Home items** category, which cannot be deleted.

You can create any number of applications and permissions that you might want to add to the **Applications** page. You can create or modify the categories to organize the **Applications** items.

For more information about customizing the **Applications** page, click ⓘ on the Dashboard.

Customizing the Look of the User Interfaces

The Dashboard allows you to modify the product title, logo, and colors in the header to meet your organization's branding requirements. In the footer you can customize with copyright/Trademark and contact information. The User Application provides a set of built-in themes that you can apply to the look and feel of the user interface. You can also customize one of these themes or create your own for branding purposes.

- "Applying Your Organization's Brand to the Dashboard" on page 112
- "Applying a Cascading Style Sheet to the Dashboard" on page 113

Applying Your Organization's Brand to the Dashboard

You can use your organization's logo and name in the header and footer of the Dashboard. You also can specify your brand colors and localize the content in the header. For more information about applying your brand to the Dashboard, click ⓘ on the Dashboard and see *Client Customization* > *Customize the User Interface* > *Customize the Branding*.

Applying a Cascading Style Sheet to the Dashboard

You can change the appearance of the Dashboard user interface by using your own cascading style sheet (css). The `idmdash.war` supports customization through a file called `idmdashcustom.css`. It looks for this file in a directory called `netiq_custom_css` within the home directory of the user that started Tomcat on the server where the application server is running.

By default, this user is `novlua`, so the home directory is `/home/users/novlua/`. If a `idmdashcustom.css` file exists, it overrides the default style sheet file provided with the Dashboard.

1. Create a new directory in the home directory of the Tomcat user running the server.
2. Add your `idmdashcustom.css` file to the `netiq_custom_css` folder created in Step 1.
3. If Tomcat is already running, refresh your browser to see the changes. Otherwise, restart Tomcat and clear the cache from your browser.

Localizing the Text in the Interfaces

The text displayed in the identity applications is stored in either a set of language-based JSON files, language-based JAR files, or properties files located in the User Application WAR and User Application driver. In general, the file name includes a reference to the language. For example, the English language strings for the User Application are stored in the `UserAppStrings_en.JAR`.

NOTE: The labels and string text typically change between versions. This means that you have to apply your string changes or customizations to each new release.

- "Localizing the Labels in the Dashboard" on page 113
- "Hiding the Applications Tab" on page 115
- "Modifying the Text of the Application Tab" on page 115
- "Localizing the Text Stored in the JAR Files" on page 116

You can also translate or localize the names and descriptions of provisioning objects in the Directory Abstraction Layer, Provisioning Request Definition, and Role Catalog. For more information, see Localizing Provisioning Objects.

Localizing the Labels in the Dashboard

Some organizations might want to customize the default names for the fields and navigation items in the Dashboard. This procedure describes the process for updating the downloadable `.properties` files.

WARNING: Do not modify any text in the code string before the = sign. For example, `category-featured-47-name =`. The Dashboard might not function appropriately if you change the code string incorrectly.

---

1. Log in to the Identity Manager Dashboard.
2. Select **Applications >** ⚙.

3 On the **Manage Applications** page, select the **Localize** icon.

The Dashboard lists the `.properties` files by language.

Language	Locale Designation
Chinese (China)	zh_CN
Chinese (Taiwan)	zh_TW
Danish	da
Dutch	nl
English	en
French	fr
German	de
Italian	it
Japanese	ja
Portuguese	pt
Russian	ru
Spanish	es
Swedish	sv

4 In the **Languages** window, download the `.properties` file for each language that you want to localize.

Depending on your browser settings, you might be prompted for the download path.

NOTE: If prompted, do not rename the `.properties` file. The Dashboard cannot upload a file that does not match the expected name.

5 In a text editor, customize the displayed text for the attributes that you want to change.

For example, if you download the `pt.properties` file to localize the Dashboard in Swedish, modify the properties file as follows:

```
# English value: My Category
category-featured-47-name = Min kategori
```

NOTE: If you want to use double-byte or extended characters in the properties file, ensure that you save the file using the correct encoding.

6 Save and close the file.

7 In the **Languages** window, upload the modified file to the appropriate language.

8 Close the **Languages** window, then select **Edit Done**.

9 Refresh the browser window to view the changes.

NOTE: Depending on the browser settings, you might need to log out of the Dashboard, clear the cache in the browser, then log in again.

Hiding the Applications Tab

1 Navigate to the `/opt/netiq/idm/apps/tomcat/conf/clients/` directory.

2 Edit the `1.json` file and ensure that the **isDisabled** value is set to **False**.

If the following section is not there, add it:

```
{
      "key": "main",
      "value": [],
      "type": "navItem",
      "isDisabled": false,
      "sectionKey": "Configuration",
      "areaDefault": false,
      "disableAreaDefault": false,
      "displayLable": "Application",
      "sectionLable": "Configuration",
      "page": null,
      "expanded": false,
      "level": 1
}
```

3 Save the `1.json` file.

4 Assign the required permissions for the `1.json` file.

```
chown novlua:novlua /opt/netiq/idm/apps/tomcat/conf/clients/1.json
```

5 Restart Tomcat.

```
systemctl restart netiq-tomcat
```

6 Log in to Identity Manager Dashboard.

7 Navigate to **Settings** > **Access**.

8 Add the required trustees.

Modifying the Text of the Application Tab

You can modify the menu text and the header on the Application tab.

To change the Application menu text:

1 Log in to the Identity Manager Dashboard.

2 Select **Applications** > .

3 On the **Manage Applications** page, select .

4 On the **Languages** page, click next to the language file you want to download. For example, `English`.

5 Modify the following line in the downloaded file. For example `en.properties`.

   `label-menu-main = Application`

   For example, you can modify the above line to:

   `label-menu-main = Applications Menu`

6 Save the `en.properties` file.

7 Click  next to English to upload the `en.properties` file.

8 Refresh the browser.

To change the header on the Applications Page:

1 Navigate to the `/opt/netiq/idm/apps/tomcat/webapps/idmdash/i18n/json/` directory.

2 Edit the `DashStringsRsrc_<language>.json` file, where `<language>` is the language file that you want to modify. For example, `DashStringsRsrc_en.json`.

3 Modify the following line:

   `"Applications": "Applications",`

   For example, you can modify the above line to:

   `"Applications": Applications List",`

4 Refresh the browser.

Localizing the Text Stored in the JAR Files

The text strings stored in the WAR files are found in a language-based `.jar` file. These `.jar` files include various `.properties` files, which you can customize for the RBPM Configuration Utility as well as most of the Dashboard and User Application interfaces. You can also customize the labels for One SSO Provider (OSP).

WARNING: Do not modify any text in the code string before the = sign. For example, `category-featured-47-name =`. The identity applications might not function appropriately if you change the code string incorrectly.

The `.jar` files are located by default in the following directories.

- **Identity applications**: `/opt/netiq/idm/apps/UserApplication/l10n-resources/userapp`

  For example, `UserAppStrings_<locale>.jar`.

- **OSP**: `/opt/netiq/idm/apps/osp/osp-extras/l10n-resources`

  For example, `osp-custom-resource.jar`.

Customizing Strings for Identity Applications

To customize strings for identity applications:

1 Log in to the server where you installed the identity applications.

2 Identify the WAR file for the identity application that you want to modify:

IDMProv.war for the User Application and Dashboard

3 In the WAR file, locate the .jar file(s) that you want to update.

For example, the UserAppStrings.JAR file contains displayed text for the User Application.

4 Copy the .jar files that you want to update to a temporary directory.

WARNING: Do not change the file names or directory structure of the .jar files.

---

5 To access the .properties files in each .jar file in the temporary directory, complete one of the following actions:

- Extract the .properties files
- Use WinRAR to open each .properties file

For example, access the OAuthManagerRsrc_en.properties file in the UserAppStrings_en.JAR.

6 Browse the file directory to the .properties file that you want to edit.

For example, UserAppStrings_fr.properties.

7 In a text editor, customize the displayed text for the content that you want to change.

WARNING: Do not modify any text in the code string before the = sign. For example, ADMIN_PASSWORD=. The identity applications might not function appropriately if you change the code string incorrectly.

---

8 Save and close the editor.

9 To apply your changes to the application, complete the following steps:

WARNING: Do not change the file names or directory structure of the .jar and WAR files.

9a Using the Java JDK jar program, add the properties files back to the .jar file.

9b Add the modified .jar to the appropriate WAR file, maintaining the folder location within the WAR.

You can use the Java JDK Jar program. For example:

```
jar -uvf IDMPRov.WAR WEB-INF/lib/UserAppStrings_fr.jar
```

9c Redeploy the WAR file to your application server.

10 Stop Tomcat.

For example:

```
systemctl stop netiq-tomcat
```

11 Delete all files and folders in the following directories:

- Tomcat temporary directory, located by default in /opt/netiq/idm/apps/tomcat/temp
- Catalina directory, located by default in /opt/netiq/idm/apps/tomcat/work/Catalina

12 Delete all log files from the tomcat/logs directory, located by default in /opt/netiq/idm/apps/tomcat/logs.

13 Start Tomcat.

For example:

```
systemctl start netiq-tomcat
```

14 Before logging in to the identity applications, clear the browser cache to ensure that the browser displays your changes.

15 To test your changes, complete the following steps:

15a Access the identity application that you modified.

15b Using your list of changes, review each occurrence of the string you changed to determine if you made the change appropriately.

Customizing Strings for OSP

To customize the strings on the OSP login page:

1 Log in to the server where OSP is installed.

2 Navigate to `osp-extras/l10n-resources`.

By default, it is located in `/opt/netiq/idm/apps/osp/osp-extras/l10n-resources`.

3 Back up `osp-custom-resource.jar` file.

4 Copy the backed up `osp-custom-resource.jar` to a temporary directory.

5 Extract the `osp-custom-resource.jar` file in the temporary directory.

For example: `jar xf osp-custom-resource.jar`

NOTE: Make sure that you have maintained the existing directory structure during extraction.

---

6 Navigate to `resources` folder and open `oidp_enduser_custom_resources_en_US.properties` file and uncomment the following properties:

WARNING: Do not modify any text in the code string before the = sign. For example, `ADMIN_PASSWORD=`. The identity applications might not function appropriately if you change the code string incorrectly.

```
## Organization name [nbsp], [reg], [tm], [amp], [br], [plus], [apos]
are pseudo-tags
## that are converted at runtime into appropriate HTML.
OIDPENDUSER.LoginProductName=Company[nbsp]Name[reg]

## Whether the company or product name should be displayed in the login
pages: "true" or "false"
OIDPENDUSER.LoginProductNameDisplay=true
```

These properties modify the banner name on the login page. You can uncomment other properties and change them to localize different texts on the login page.

7 Save and close the editor.

8 Stop Tomcat.

For example:

```
systemctl stop netiq-tomcat
```

9 To apply your changes to the application, complete the following steps:

WARNING: Do not change the file names or directory structure of `.jar` and WAR files.

9a Using the Java JDK jar program, add the properties files back to the `.jar` file.

9b Update the `osp-custom-resource.jar` with the customized properties files in the temporary directory.

You can use the Java JDK Jar program. For example:

```
jar -uf osp-custom-resource.jar resources/
oidp_enduser_custom_resources_en_US.properties
```

9c Copy the updated `osp-custom-resource.jar` to the `tomcat/lib` directory.

10 Delete all files and folders in the following directories:

- Tomcat temporary directory, located by default in `/opt/netiq/idm/apps/tomcat/temp`
- Catalina directory, located by default in `/opt/netiq/idm/apps/tomcat/work/Catalina`

11 Start Tomcat.

For example:

```
systemctl start netiq-tomcat
```

12 Before logging in to the identity applications, clear the browser cache to ensure that the browser displays your changes.

Adding a Language to the Identity Applications

If the default languages for Identity Manager do not meet your organization's needs, you can translate the strings and user interface content to a different language. For example, you might want to interact with the identity applications in Norwegian (language code=nb). To translate content to a non-default language, you can copy the `.properties` files of an existing language.

To complete this process, we recommend perform the following steps in the listed order.

	Checklist Items
❑	1. Configure the identity applications to support the new language. For more information, see "Adding the New Language to the Identity Applications" on page 120.
❑	2. Copy an existing set of files that you can use as a template for translating to the new language. For more information, see "Preparing Files for Translation" on page 121.
❑	3. Translate the files.
❑	4. Change the default language to the new language. For more information, see "Changing the Default Language" on page 123.
❑	5. Add the translated files to the appropriate locations, such as WAR files or upload to the user interface. For more information, see "Add the Translated Files to the Proper Locations" on page 124.

	Checklist Items
❑	6. Update notification templates using designer. For more information, see "Updating an Email Notification Template" on page 125.
❑	7. Verify that the identity applications display the appropriate content. For more information, see "Verifying the New Translations" on page 125.

For more information about customizing the content for a existing language, see "Localizing the Text in the Interfaces" on page 113.

Adding the New Language to the Identity Applications

You must configure the identity applications to support the new language. You can perform this action in iManager.

1. Log in to iManager as an Administrator.
2. Click icon for **View Objects**.
3. In the tree, navigate to ***Context*** **>** ***Driver Set*** **>** ***Driver*** **> AppConfig > AppDefs**.

   For example, **netiq > TestDrivers > UserAppDriver > AppConfig > AppDefs**.
4. Click **locale-configuration**.
5. In the **Valued Attributes** list, select **XmlData**, then click **Edit**.
6. In the **Edit Attribute** window, search for `<locale code"=xx> </locale>` and create similar tags with the values for the language that you want to support.

   Ensure that you verify the language code.
7. Search `<supported-locale>xx</supported-locale>` and add a new tag with the newly created language code. For example:

   ```
   <supported-locale>en</supported-locale>
   ```
8. Click **OK**, then **OK** again.
9. Restart the application server.
10. To verify that the identity applications can display the new language, complete the following steps:

    10a Log in to the Dashboard as an administrator.

    10b Select **Applications >** ⚙.

    10c On the **Manage Applications** page, select the **Localize** icon.

    The Dashboard lists the `.properties` files by language.

    10d Verify that the language you added to iManager appears in the list of Languages.

    Although the Dashboard displays a Download option for the new language, there is not content to download. To create that content, continue to "Preparing Files for Translation" on page 121.

Preparing Files for Translation

The identity applications display content from several types of language-based `.properties` files from the following sources:

- `.json` files, such as `DashStringsRsrc_en.json`
- `.json` files, such as `AdminStringsRsrc_en.json`
- `.jar` files, such as `UserAppStrings_en.jar`

  WARNING: Do not change the directory structure of the `.jar` files or modify any text in the code strings before the = sign. The identity applications might not function if you make inappropriate alterations.

- Downloadable files, such as `localizedLabels_en.properties`

  To make it easy to modify the content that the identity applications source from the WAR files, we enable you to download the `.properties` files directly from the Dashboard. You do not need to edit the WAR files.

You use different tools to customize the text depending on where the text is stored. To ensure that all content appears in your preferred language, you must translate all of the files. This procedure assumes that you will translate English `.properties` files to the new language, rather than starting from another language such as French.

For more information about customizing the interface content, see "Localizing the Text in the Interfaces" on page 113.

To prepare files for translation:

1. Complete the steps in "Adding the New Language to the Identity Applications" on page 120.
2. To prepare the file that the Dashboard uses for labels in the user interface, complete the following steps:
   - **2a** To download a file to use as the template for translation, complete the procedure in "Localizing the Labels in the Dashboard" on page 113.
   - **2b** Change the locale code in the file name to represent the language that you want to add.

     For example, to add Norwegian, change

     `localizedLabels_en.properties`

     to

     `localizedLabels_nb.properties`
3. To prepare the content in the `.jar` files, complete the following steps:
   - **3a** Create backup copies of the `.jar` files that you want to translate. Store the backups in a safe location.

     For more information about updating `.jar` files, see "Localizing the Text Stored in the JAR Files" on page 116.
   - **3b** Copy the `.jar` files that you want to translate to a temporary directory.

     You will need these files again after the translations are complete.

3c To access the `.properties` files in each `.jar` file in the temporary directory, complete one of the following actions:

- Extract the English `.properties` files
- Open the properties file in WinRAR

For example, access the `OAuthManagerRsrc_en.properties` file in the `UserAppStrings_en.JAR`.

3d For each `.properties` file, change the locale code in the file name to represent the language that you want to add.

For example, to add Norwegian, change

```
OAuthManagerRsrc_en.properties
```

to

```
OAuthManagerRsrc_nb.properties
```

3e Within each `.properties` file, change the language code in the key `BUNDLE_LOCALE` to represent the language that you want to add.

For example, to add Norwegian, change

```
BUNDLE_LOCALE=en
```

to

```
BUNDLE_LOCALE=nb
```

3f (Conditional) If a string that you want to translate and use in the `.properties` file has a comment, you must un-comment it.

For example, change

```
#OIDPENDUSER.50048=Next
```

to

```
OIDPENDUSER.50048=Next
```

3g Create `.jar` files to contain the `.properties` files that you want to translate.

For example, for the Norwegian translator, you might create `UserAppStrings_nb.jar`.

The new `.jar` files must mimic the directory structure of the original files.

3h Add the `.properties` files that are ready for translation to the new, appropriate `.jar` files.

For example, add the `OAuthManagerRsrc_nb.properties` file to the `UserAppStrings_nb.jar` file.

4 To prepare the content in the .json files, complete the following steps:

4a Locate the files as described in "Localizing the Text Stored in the JAR Files" on page 116.

5 Provide the `.jar` files, `localizedLabels_xx.properties` files, and `.json` files to your translator.

WARNING: Ensure that the translator maintains the file names and directory structure of the `.jar` files. Also, do not modify any text in the code string before the = sign. For example, `com.netiq.UA.persistence.ops.AttributeDefinition.USER.guid=`. The identity applications might not function if you make inappropriate alterations.

---

6 To add the supported locale for `idmadmin.war`, perform the following steps:

6a Navigate to `idmadmin.war` and open this file using Winrar.

```
\opt\netiq\idm\apps\tomcat\webapps\idmadmin.war
```

6b Add the new `AdminStringsRsrc_xx.json` file and validate this file.

```
\idmadmin.war\assets\i18n\
```

7 To change the supported locale for `idmadmin.war`, perform the following steps:

7a Navigate to `idmadmin.war` and open this file using Winrar.

```
\opt\netiq\idm\apps\tomcat\webapps\idmadmin.war
```

7b Edit the `AdminStringsRsrc_xx.json` file and validate this file.

```
\idmadmin.war\assets\i18n\
```

7c Copy the edited `AdminStringsRsrc_xx.json` file to the following location.

```
\idmadmin.war\assets\i18n\
```

Changing the Default Language

The `configupdate` Utility controls which languages appear in the identity applications and sets the default language. Perform this procedure when you are ready to add new translations to the identity applications.

1 Complete the steps in "Preparing Files for Translation" on page 121.

2 In a terminal, navigate to the `/bin` directory, located by default in `/opt/netiq/idm/apps/UserApplication` or `C:\NetIQ\idm\apps\UserApplication`.

3 At the command prompt, use one of the following methods to run the configuration utility:

- **Linux:** `./bin/configupdate.sh`
- **Windows:** `configupdate.bat`

NOTE: You might need to wait a few minutes for the utility to start up.

---

4 Select **Miscellaneous**.

5 For **Supported Locales**, add the locale code that represents the language(s) that you want to include. Use a pipe sign to separate entries.

For example, enter `|nb` for Norwegian.

6 For Default Locale, specify the language that you want to use.

   For example, enter nb for Norwegian.

7 Save your changes and close the utility.

Add the Translated Files to the Proper Locations

After translations are completed, you can add the translated files to their appropriate WAR and uploaded locations.

1 Log in to the server where you installed the identity applications.

2 Copy the `.jar` file(s) to `WEB-INF/lib/`, by default in the `/opt/netiq/idm/apps/tomcat/webapps/IDMProv/` directory.

3 Upload the `.json` file to `/opt/netiq/idm/apps/novlua/netiq_custom_css`.

4 Stop Tomcat.

   For example:

   ```
   systemctl stop netiq-tomcat
   ```

5 Delete all files and folders in the following Tomcat directories:

   - `temp`, located by default in `/opt/netiq/idm/apps/tomcat`
   - `Catalina`, located by default in `/opt/netiq/idm/apps/tomcat/work`

6 Delete all log files from the Tomcat `logs` directory, located by default in `/opt/netiq/idm/apps/tomcat`.

7 Start Tomcat.

   For example:

   ```
   systemctl start netiq-tomcat
   ```

8 To upload the label files, complete the following steps:

   8a Log in to the Dashboard as an administrator.

   8b Select **Applications >** ⚙.

   8c On the **Manage Applications** page, select the **Localize** icon.

   8d For the language that you added to the identity applications, select **Upload**.

   For example, if you added the locale code for Norwegian, upload the `localizedLabels_nb.properties` file.

   8e Refresh the browser window to view the changes.

   NOTE: Depending on the browser settings, you might need to log out, clear the cache in the browser, then log in again.

Updating an Email Notification Template

After adding a language to the identity applications, update the notification templates for the newly added language using Designer.

1. Ensure that your localized notification template includes an appropriate locale code in the filename.

   If you are updating the localized template for Norwegian language, ensure your template's fiename includes `.nb`. For example `Email Based Approval Templates.nb`

2. Import the modified notification template into **Default Notification Collection**.

   2a Right-click **Default Notification Collection** and select **Import Templates from File**.

   2b Browse to and select the notification template.

3. Right-click **Default Notification Template** and select **Live > Deploy**.

Verifying the New Translations

1. In a browser, clear the browser cache.
2. Change the browser language to the language that you added.
3. Enter the URL for the identity applications.

   If you did not translate the content in the OSP `.jar` files, the login page continues to appear in the default language.

4. Log in.
5. Observe the translated content.

Configuring User Names

The Dashboard and the User Application allow you to configure the format of displayed user names in your environment based on the user's current locale. To simply name entry, the identity applications attempt to complete the names as you type them based on the information in the database.

- "Configuring the Format of Displayed User Names" on page 126
- "Enabling Localized User Names in Typeahead Fields" on page 126

Configuring the Format of Displayed User Names

The Dashboard and the User Application allow you to configure the format of displayed user names in your environment based on the user's current locale.

You can then use localized user names in Approval forms, using the literal `%LocaleFormattedFullName%` for forms with the `User` entity definition key. For more information about creating or configuring forms in Designer, see "Creating Forms for a Provisioning Request Definition", in the *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.

1. Start Designer.
2. Open your current project and click the project name in the Outline view.
3. In the Provisioning view, right-click **Full Name** and select **Edit**.
4. In the Directory Abstraction Layer editor, expand **Entities > Full Name**.
5. Select the locale name pattern that you want to modify.
6. Modify the **Calculated Attribute** expression to specify the format you want to use for the locale. For example, if you want to display the user's surname first and given name second, modify the expression as follows:

   ```
   attr.getValue("Surname") + " " + attr.getValue("Given Name")
   ```

   You can either modify the expression manually in the Expression field or click the **Build ECMAScript Expression** icon and use the ECMA Expression Builder to modify the expression. For more information about modifying ECMAScript expressions, see "Working with ECMA Expressions" in the *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.
7. Save your changes to the locale name pattern.
8. Repeat Step 5 through Step 7 for each name pattern you want to configure.
9. When finished, close the Directory Abstraction Layer editor.
10. In the Modeler, right-click the User Application driver and select **Driver > Deploy**.
11. Click **Deploy**, then click **Yes** to restart the driver.
12. Click **OK**.

Enabling Localized User Names in Typeahead Fields

After you configure the `Full Name` entity in the Directory Abstraction Layer, the identity applications automatically display user names formatted by locale.

However, to use user names with localized name formatting in typeahead fields within the identity applications, you must create one or more custom registry entries. The identity applications use typeahead controls when a supervisor wants to manage a specific or team, and the typeahead controls do not use the `%LocaleFormattedFullName%` literal.

1. In the `conf` directory of your application server installation, create an empty file with the file name `UIControlRegistry_CustomProps.xml`.
2. Open the User Application WAR, `IDMProv.war` by default, and extract the contents.
3. Locate the `UIControlRegistry.xml` file in the WAR's `WEB-INF` directory.

4 In the UIControlRegistry.xml file, locate the entries for the UserDNLookup and UserInTeamDNLookup keys.

5 Copy the <registry> element, <ctrls> element, and both keys to the UIControlRegistry_CustomProps.xml file.

6 Modify the display-exp property of each of the copied keys as follows:

```
<prop name="display-exp" type="string">
    <value>FirstName LastName</value>
    <value xml:lang="en">LastName FirstName</value>
</prop>
```

7 Create an xml:lang value for each localized name format you want to use. You can include a value for each language supported by the User Application.

8 Save and close the UIControlRegistry_CustomProps.xml file.

9 Restart Tomcat.

Configuring Email Notification Templates for the Dashboard

By default, email notification templates in Identity Manager direct recipients to the User Application. To direct recipients to the Dashboard, you must modify the default email notification templates in Designer.

NOTE

- Not all notification templates include links to the User Application.
- Modifying an existing notification template marks that template as customized in Designer.

To direct recipients to the Dashboard:

1 Start Designer.

2 Make sure you have imported email notification templates into your Designer project.

3 In the **Outline** view, right-click the notification template you want to modify and select **Copy**.

NOTE: We recommend you create and modify a copy of the original notification template you want to configure, rather than modifying the original. You can then specify the “Identity Manager Dashboard” version of the template in any workflows where you want users to use the Dashboard, and not modify the workflows where you want users to use the User Application.

---

4 Specify a name for the copied template and click **OK**.

5 Right-click the copied template and select **Edit**, then click **Yes** to confirm.

6 (Optional)To remove all links to the User Application, change all instances of the following text:

```
$PROTOCOL$://$HOST$:$PORT$/$TASKLIST_CONTEXT$
```

to:

```
$PROTOCOL$://$HOST$:$PORT$/idmdash/#tasks
```

7 (Optional) To retain the existing User Application links, add text similar to the following line to the notification template message:

```
You can review your tasks list using the Dashboard at
$PROTOCOL$://$HOST$:$PORT$/idmdash/#tasks.
```

8 When finished, save and close the notification template.

9 Repeat Step 3 through Step 8 for each notification template you want to modify, including any localized templates.

10 Deploy the new templates to the Identity Vault.

11 Modify any workflows where the approver should use the Dashboard so the workflow uses the new notification templates.

Configuring Forgot Password? Functionality

If you want the Dashboard login page to display the **Forgot password?** link, you must configure the Password Management Settings in the **Authentication** tab of the `configupdate` utility and then restart the Tomcat application server.

1 Launch the `configupdate` utility from the command line: `configupdate.sh` or `configupdate.bat`

2 Navigate to **Authentication > Password Management**, then select **Forgot Password**.

3 Click **OK**.

4 Click **Logout**.

5 To start Tomcat, enter the following command in a command prompt:

```
/etc/init.d/netiq-tomcat restart
```

6 After Tomcat finishes restarting, go to the Dashboard login page to verify the page displays the **Forgot password?** link.

Ensuring that Characters Display Properly in Role Report PDF Files

By default, the role report feature of the identity applications uses “UniGB-UCS2-H” for the PDF encoding and “STSong-Light” for the PDF font for Chinese simplified, Chinese traditional, Russian and Japanese locales. For the other locales, “Cp1252” is used for PDF encoding and “Helvetica” or “Helvetica-Bold” is used for the PDF font.

If the user's browser locale or preferred locale is set to one of the above four locales, the report will be able to display most of characters from these locales. However, some extended characters found in ISO-8859 may not be displayed properly in the report.

Conversely, if the browser locale or preferred locale is not set to one of these four locales then some Asian characters will not display properly.

To allow all characters to display properly in generated PDF files, you need to:

- Edit the Configuration XML Data in iManager
- Configure the identity applications

NOTE: You may also notice problems displaying some characters in role reports for languages that are not in the standard set of supported languages. If you add a new language (such as Polish), you may also need to perform the steps provided in this section to ensure that all characters display properly for that language as well.

- "Editing the Configuration XML Data in iManager" on page 129

Editing the Configuration XML Data in iManager

1. Login to iManager as your Administrator.
2. Click the View Objects icon.
3. In the Tree, navigate to the following location:

   Context > *Driver Set* > *Driver* > AppConfig > AppDefs

   For example:

   netiq > TestDrivers > UserAppDriver > AppConfig > AppDefs

4. Click **configuration**.
5. In the Valued Attributes list, select XmlData and click **Edit**.
6. In the Edit Attribute window, search for PREF_FONT and replace the corresponding <value></value> with <value>Arialuni.ttf</value>.
7. Search for PREF_ENCODING and replace the corresponding <value></value> with <value>Identity-H</value>.
8. Click **OK**, the click **OK** again.
9. Restart the User Application driver.

Ensuring that Dates Display Correctly in Norwegian

For language codes no and nb, you need to perform a workaround to ensure that dates display correctly in Norwegian. The Date.js file contains no but not nb, however, the dmask value (dd/MM/yyyy) is not correct. For both no and nb, the format should be dd.MM.yyyy.

To ensure that dates display correctly in Norwegian:

1. Copy the file com/netiq/common/i18n/I18nDateTimeRsrc_en.properties, modifying the locale portion of the file name to match the desired locale (for example, I18nDateTimeRsrc_nb.properties).
2. Modify the format(s) in the file to match the desired format. There are four format types: short, medium, long and full. These formats correspond to the java.text.DateFormat.SHORT, .MEDIUM, .LONG and .FULL constants.

3 Add the file to the IDMProv.war under WEB-INF/classes/com/netiq/common/i18n using the jar utility (file must be placed in a directory tree corresponding to the above path).

```
jar uvf IDMProv.war
WEB-INF/classes/com/netiq/common/i18n/I18nDateTimeRsrc_nb.properties
```

Configuring Client Settings Mode

Administrator with the settings page navigation rights can configure multiple clients with different settings using the Identity Manager dashboard. Administrator can decide to save these configurations using the following modes as part of application configuration:

Using File System

The client settings directory is stored in the **<tomcat base folder>/conf** folder, by default the directory is not configured. The clients settings configuration can also be stored in a clients settings directory under the **User Home** folder. The administrator needs to add the `com.netiq.idmdash.client.settings.directory` in the `ism-configuration property` file with the appropriate value.

NOTE: You can set client settings directory as %`user.home`% to create the client settings directory under the **User Home** folder. You can also set the client setting directory as %`catalina.base`% to create the client setting folder under **<tomcat base folder>/conf**. If you do not set the client settings directory to any of the above mentioned values, the directory will be created under the **User Home** folder.

Using Database

Administrator can also store the configuration in the `CLIENT_SETTINGS` table in the Identity User Application database.

By default the client settings configuration is stored in File System mode. The administrator can specify the mode of saving the configuration in the `ism-configuration properties` file. You can add the `com.netiq.idmdash.client.settings.store.preference` in the `ism-configuration properties` file with the appropriate value. If you want to change the mode, the property value should be set to `database`.

IMPORTANT

- Always select the `database` mode for saving the client settings configuration on each node of the cluster environment.
- If you want to migrate from file system to database mode or vice versa, client settings do not get migrated. Identity Manager Dashboard reflects the settings which is already configured in the selected mode.

Changing Identity Applications Client Settings

Identity Manager Dashboard allows you to modify the settings for every client configured in identity applications. An administrator must have access to **Your ID > Settings** page to modify the client settings.

The **Settings** page also allow you to add a client settings in the identity applications. To add a new client settings, click + and specify the **Client Name** and **LDAP Filter to match**. You can perform the following activities in this page:

- "Changing General Client Settings" on page 131
- "Customizing the Views" on page 131
- "Managing User Access" on page 135
- "Changing the Client Branding Attributes" on page 136
- "Configuring a Client Helpdesk" on page 136
- "Managing Dashboard Widgets" on page 139
- "Deleting the Client Settings from Identity Applications" on page 139

Changing General Client Settings

The **General** tab allows you modify the basic client settings.

Client Name

Represents the name of a client.

LDAP Filter to match

Represents the condition that helps to determine the client settings to be applied for the logged in user. If no condition is matching then it applies the default client settings.

Customizing the Views

The **Customization** tab allows you to modify the **Users** page view for the selected client. You can also specify general settings for notifications and request forms.

Select the following options from **Navigation items** to customize views for your client:

- "User Settings" on page 131
- "General Settings" on page 133
- "Entity Settings" on page 133

User Settings

The **User** settings enable you to configure the attributes displayed in the **Users** page for the selected client.

Card View

Represents the attributes that you want the application to display by default when the user selects **Card View** in the **Users** page.

Other Attributes

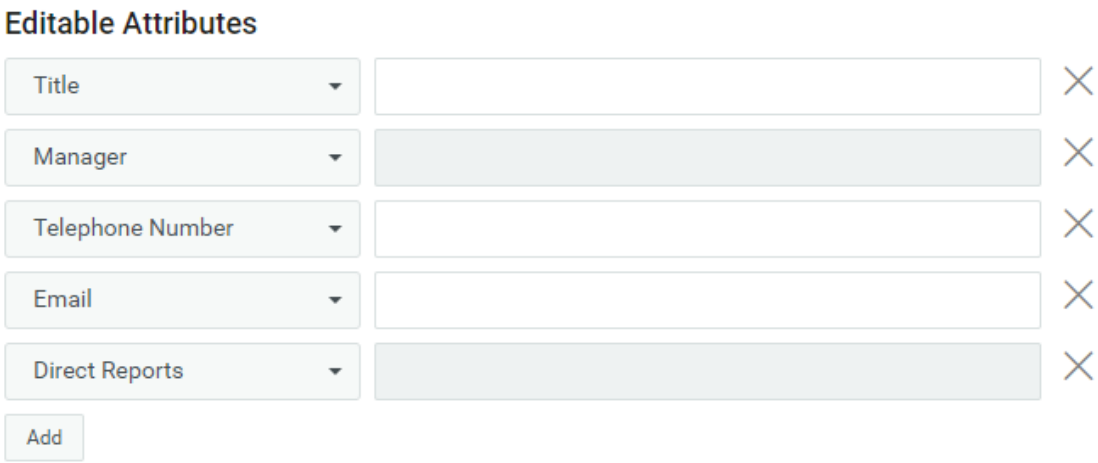
Represents additional attributes that provide details about a selected user.

Editable Attributes

Represents the attributes that can be modified for a user's details. For most attributes, you can also enter text to serve as default values or examples to aid in new user creation, as desired.

The following example allows a client user to edit Title, Manager, Telephone Number, Manager, and Direct Report attributes:

Figure 10-1 *Editable Attributes*



To add more attributes to the list, click **Add**. You can also select a different attribute from the list to modify the editable attributes.

User Search Lookup Attributes

Represents the attributes that users can define when searching for a user or filtering search results in the **Users** page.

User General Settings

Represents the default container for storing users and how the application responds when displaying search results.

- **Base Container**

  Specifies the container in the Identity Vault that stores a newly created user.

  When creating a user, you can see this value but cannot modify it. This limitation ensures that all users are stored in the same container for that client.

- **User Search Limit**

  Specifies the maximum number of users that the application can list as a result of a user search.

- **View Permission Type**

  Enable the permission types such as **Roles**, **Resources**, and **PRD**. This allows your client users to view or request the permission types that are selected.

  By default, all the permission types are enabled.

General Settings

The **General** settings specify how the client responds upon user login and when the user initiates forms.

Notification Expiry

Specifies the number of days before a task or role expires that the application begins displaying a notification when the user logs in.

Open Request Form in a New Window

Specifies whether the application opens a new window or a dialog box when the user makes a new request for permissions. When selected, the application opens a new window.

Open Approval Form in a New Window

Specifies whether the application opens a new window or a dialog box when the user selects task to approve. When selected, the application opens a new window.

Enable Task Bulk Approval

Allows the client users to approve or deny multiple requests at a time.

Set Availability while creating a Delegation Assignment

Specifies whether the application displays options for providing the availability details when the user creates a delegation. When selected, the application displays the availability options at the same time when delegation is created. If you want to create delegation and specify availability details in separate actions, do not select this option.

Feedback Message Span

Specifies the period for an information message to appear on the page.

Identity Governance URL

Specifies the Identity Governance URL.

Managers Hierarchy

Specifies the manager's hierarchy. This helps the helpdesk users to reassign the helpdesk tickets to the managers of the specified level. You can set the hierarchy up to 3.

Enable Eager Search Results in Roles and Resources Page

Enable this option to display the roles in the Roles page and the resources in the Resources page. By default, this option is enabled.

Disabling this option will not display the roles and resources when the Roles and the Resources pages are loaded.

Entity Settings

The Entity settings enable you to configure the attributes. The entities are created using Designer. For more information see, About Entities and Attributes in the *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.

To configure an entity, click the + icon and select the entity type from the drop down menu. The deployed entities are displayed in the drop down menu. Click **Create**. This displays the selected entity in the **Entities** tab and in the **Navigation items** menu.

To delete an entity, select the entity from the **Navigation items** menu and click the icon. The deleted entity will not be listed under the **Entities** tab.

View Attributes

Represents the attributes that you want the application to display by default when you select the created entity in the **Entities** tab

In the example shown in Figure 10-2 on page 134, description, direct reports, company, language, photo, manager, city, mobile, title, and CN are selected for display by default.

Figure 10-2 *View Attributes*

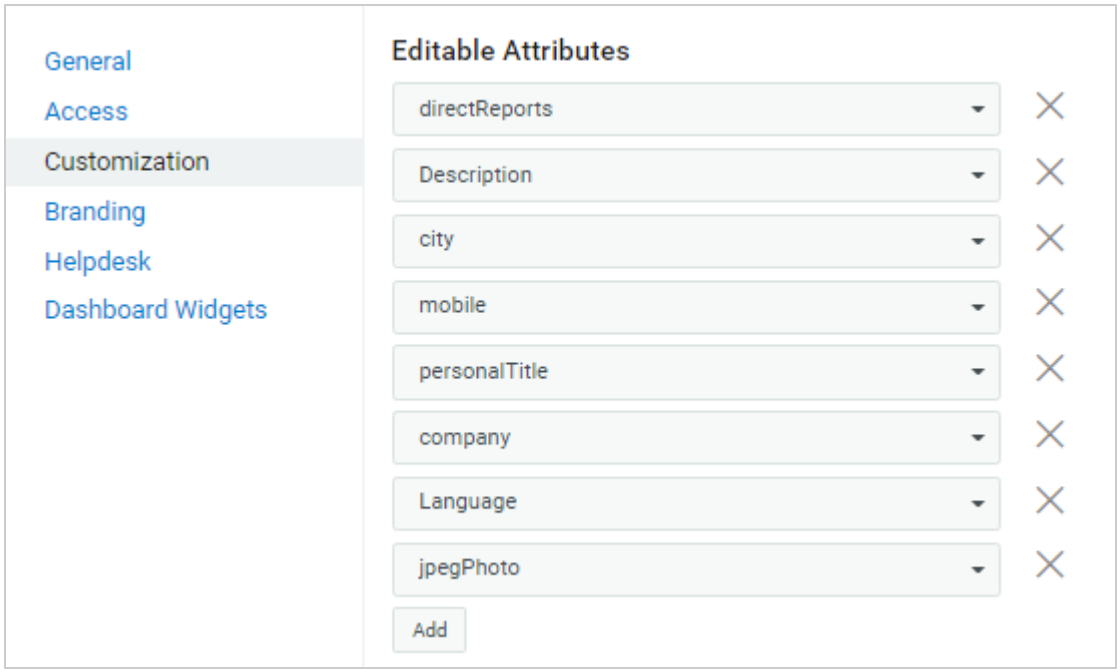


Editable Attributes

Represents the attributes that can be modified for an entity. You can add or delete attributes from the list of available entities.

In the example shown in Figure 10-3 on page 134, direct reports, description, city, company, language, photo, manager, mobile, and title attributes are editable.

Figure 10-3 *Editable Attributes*



To add additional attributes to the list, click **Add**.

Search Attribute

It represents the attributes that can be used to search an object. This setting is used to list the entities. This is a mandatory attribute.

In the example shown in Figure 10-4 on page 135, CN and Description are the search attributes.

Figure 10-4 *Search Attribute*



Base Container

Specifies the container in the Identity Vault that stores a newly created entity.

This value cannot be modified. Therefore, all entities are stored in this container for a particular client.

Managing User Access

You can control user accesses using **Access** tab, this allows you to specify which user accounts are trustees for the user and configuration based functions within the client. When a trustee logs in, the application displays the page that has been provisioned. Otherwise, the page is hidden. You can add users, groups, roles, and containers as trustees.

When configuring user access, you should consider the following conditions:

- Make sure that the users specified in **Trustees** are having sufficient Identity Vault rights to perform tasks within the Identity Applications. However, trustees can access the page but operations on the page will fail if they do not have the proper Identity Vault rights.
- Each **Navigation item** has a set of default trustees suitable for the services that can be accessed through that page. However, if you remove all trustees for a navigation item, every user will be able to access that page.
- If a user does not have access to the default navigation (or to the default menu item within a navigation area), the application redirects the user to the **Dashboard** page. The application might also display an error message, such as when a user attempts to login to page without proper authorization. The user can log in but will be directed to the **Dashboard** page.
- When a user is in **proxy** mode, the application provides access according to the permissions for the account being proxied, as opposed to the permissions for the logged in user. The proxy can perform tasks on behalf of the other user but does not assume any of the role-type permissions. For example, a user cannot perform Domain Administrator functions on behalf of a Domain Administrator unless that user also has that role.

For more information, click ⓘ on the Dashboard.

Changing the Client Branding Attributes

You can use your organization's logo and name in the header and footer of the dashboard. You also can specify your brand colors and localize the content in the header.

In **Advanced Settings**, you can specify the customized cascading style sheet (CSS).

1. Click **Download Sample CSS**, to download the sample `Custom.css` file.
2. Modify the `Custom.`CSS file values and click **Upload CSS**.

For more information about applying your brand to the Dashboard, click ⓘ on the Dashboard and see *Customize the Branding*.

Configuring a Client Helpdesk

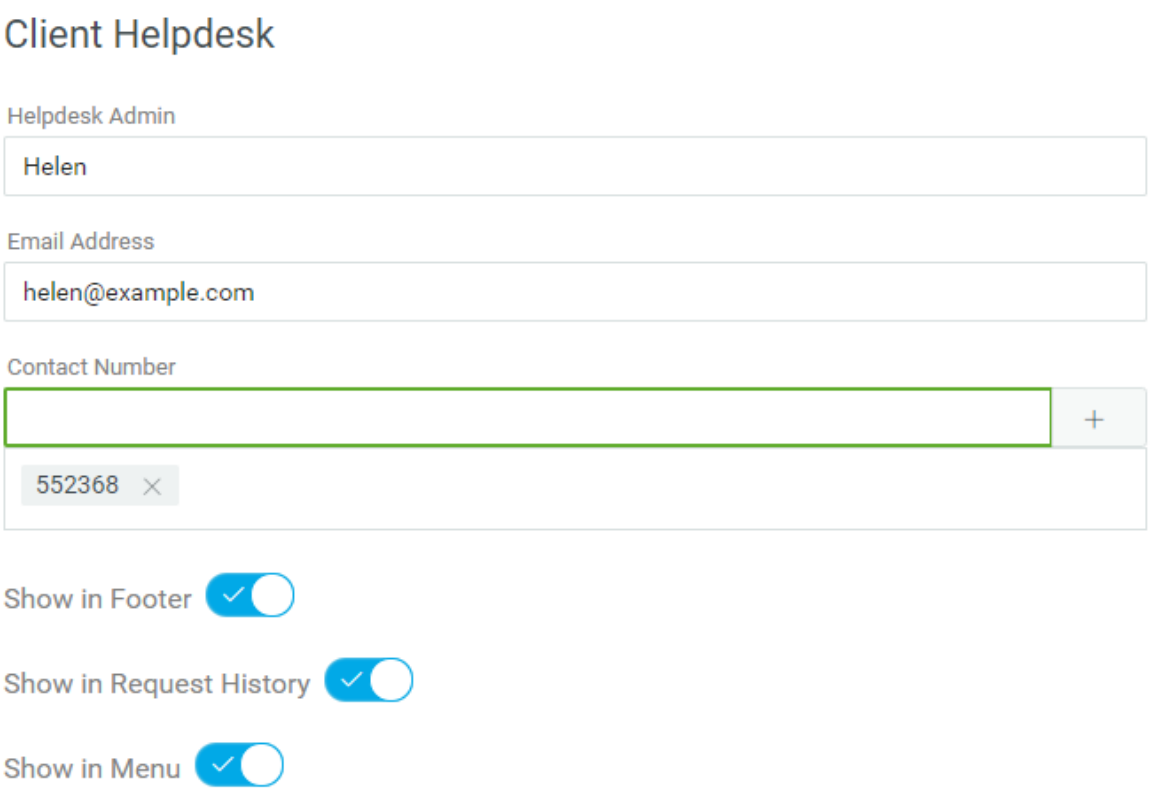
Helpdesk configuration is a two-part process. First you set up the Helpdesk contact details followed by setting up granular permission details for the Helpdesk users.

As part of configuring Helpdesk contact details, you can specify Helpdesk information such as **Helpdesk Admin**,**Email Address**, and **Contact Number** for each of the clients configured in the system. A client user will immediately see the Helpdesk contact information when the administrator completes configuring the Helpdesk for the client.

A Helpdesk user is a user configured to perform certain tasks for the tickets raised by the client users. For more information about what tasks can be performed by a Helpdesk user, see "Understanding a Client Helpdesk" on page 37.

NOTE

- Ensure that the helpdesk admin user has **read** access to **manager** property of the user container. For more information about adding user properties with required rights, see Verifying the User Properties in *NetIQ Identity Manager Setup Guide for Windows*.
- NetIQ recommends not to assign helpdesk admin role to a team manager to avoid conflicts in between these roles.

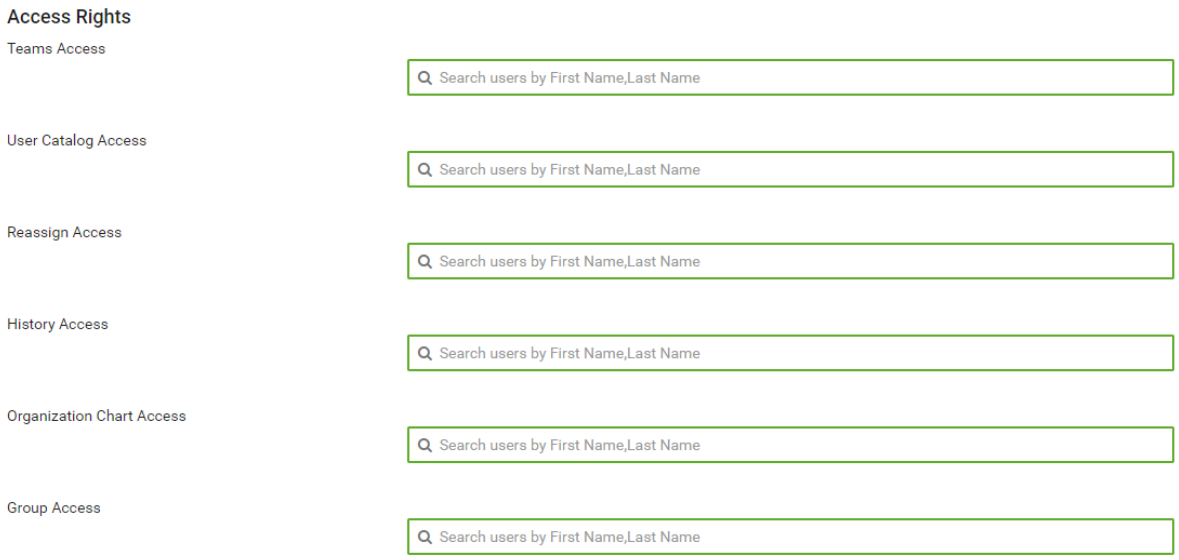


For granting permissions, you must assign Helpdesk resources to the corresponding Helpdesk users.

NOTE: You cannot assign Helpdesk resources directly to a team or group. If you want to grant helpdesk accesses to a team or group, you must include each members from the group or team individually for the required access rights. Alternatively, you can perform the following steps:

1. Create a role for helpdesk users. See "Creating a New Role" on page 154.
2. Map the required helpdesk resources to the newly created role. See "Mapping Resources to Roles" on page 156.
3. (Conditional) Assign this role to a group. See "Assigning Roles to Users" on page 157.
4. (Conditional) Team Manager can request for this role on behalf of team members. Approving this role to the team allows team members to use helpdesk resources.

The Helpdesk section lists all Helpdesk resources.



Teams Access

Selected users are allowed to view teams and team members configured for the respective client.

User Catalog Access

Selected users can view details of any user of the respective client.

Reassign Access

Selected users can reassign the user's tasks to the approver's manager.

NOTE: You can configure **Managers Hierarchy** in **Customization** to help the Helpdesk users to reassign the user's tasks to the managers of the specified level, if necessary.

History Access

Selected users can view request history of any user of the respective client.

Organization Chart Access

Selected users can view the organization chart of the respective client.

Group Access

Selected users can view groups of the respective client.

Using Helpdesk

After Helpdesk is configured, users can find the Helpdesk information in the following places:

- (Conditional) Navigate to **Your ID > Helpdesk**.

  To show the Helpdesk information in the Dashboard menu, enable the **Show in Menu** option from the **Settings** > **Helpdesk** page
- (Conditional) On the **Request History** page.

To show the Helpdesk information in the Request History page, enable the **Show in Request History** option from the **Settings** > **Helpdesk** page.

- (Conditional) At the footer.

  To show the Helpdesk information in the footer, enable the **Show in Footer** option from the **Settings** > **Helpdesk** page.

NOTE: You must ensure that the footer is enabled for the client that you have selected. To enable it, go to **Settings > Branding > Footer**. For more information, click ⓘ on the Dashboard.

Managing Dashboard Widgets

You can provision **Dashboard Widgets** for a User, Group, Container, or a Role.

You also can modify the Trustees for a selected widget. For more information, click ⓘ on the Dashboard.

Deleting the Client Settings from Identity Applications

If you want to remove a client settings from identity applications, perform the following steps:

1. Click ⚙.
2. Select the client settings from the table that you want tot delete.
3. Click 🗑.

11 Setting Up the Dashboard for Identity Applications

This section helps you to set up Identity Manager Dashboard.

Checklist for Setting Up the Dashboard for Identity Applications

NetIQ recommends that you review the following checklist for setting up the enhanced Identity Manager Dashboard:

	Checklist Items
❑	1. (Conditional) If you have installed identity applications on Linux, you must create compound indexes for all the basic attributes. To use any other attributes, you must create compound index for those attributes. For more information about creating compound indexes, see Creating Compound Indexes in the *NetIQ Identity Manager Setup Guide for Windows*. **NOTE:** If you create a compound index for a multivalued attribute and this attribute has multiple values, the identity applications return duplicate records in user catalog when you sort using that attribute. For example, if you created a compound index for multivalued attribute named as First Name, and it holds multiple values, you will see duplicate records for each values when this attribute is used for sorting. The Linux installer automatically creates the compound indexes for all the basic attributes.
❑	2. Add a new language that is not a default language, see "Adding a Language to the Identity Applications" on page 119.
❑	3. Modify the administration configuration settings for the Dashboard. You can customize the following settings: ◆ User access to pages ◆ Attributes displayed in user profiles ◆ Logo, stylesheet, and other brand settings For more information, click ⓘ on the Dashboard and see *Customize the User Interface*.

	Checklist Items
❑	4. Add links to the **Applications** page to provide your users easy access to common permissions and activities. For more information about providing links for your users, see "Linking the Dashboard to External Applications" on page 111 and click ⓘ on the Dashboard. **NOTE:** With Identity Manager 4.7, the Dashboard replaces Identity Manager Home and Provisioning Dashboard. The Dashboard's **Applications** page replicates **Featured Items** that were part of Identity Manager Home.

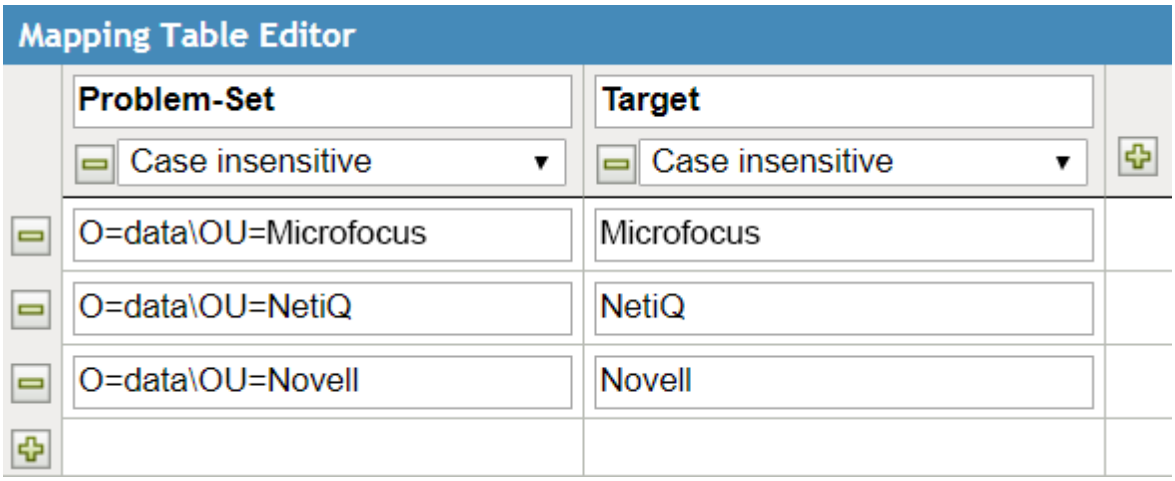
12 Configuring a Multi-Threaded Role and Resource Service Driver

This chapter provides information about setting up a Role and Resource Service driver for multi-threaded services.

How the Driver Works

In a multi-threaded driver environment, a Role and Resource Service driver uses a unique data set to achieve parallel processing of requests. The driver makes use of worker threads to process requests to unique identity data (disjoint set), where data is different based on certain attributes. This requires each driver thread to have a unique identity data, where data is different based on certain attributes. For example, Container 1 and Container 2 have different users, group, or subcontainers.

Identity Manager provides a default mapping table with two columns to support defining a unique data set based on the Organizational Unit (OU) criteria. The first column lists the criteria to search in the data set and the second column contains the name of the unique data set to search for (should be unique).



In this example, the first column contains the container DNs and the second column contains the user-defined key for a unique data set, where container DN is the criteria for identifying the data set.

By default, this mapping table is internally linked to the following Identity Manager policies:

- Resolve the disjoint set for which the role request or resource request belongs to
- Resolve the disjoint set for which the user resynchronization request belongs to

When the driver starts processing the requests, it creates a worker thread with the same name that you specified for the unique data set.

To use a different criterion for differentiating data, you must define a unique data set in the Mapping Table editor. For example, specify the container DN as a criterion for identifying a data set. After defining the data set, add the mapping table to an existing policy or create a new policy.

When the driver receives a request from the engine, it processes the associated policies and determines the unique set of the request and stores the request in the driver storage. If a worker thread is already created for the unique data set, then the driver's main thread hands over the request to that worker thread for processing. Otherwise, it dynamically creates a worker thread for that data set and hands over the request to that thread. As request processing is actually done by the worker threads, the main thread is free to receive new requests from the engine

When the driver completes processing a request, it removes that request from the driver storage. If the driver stops abruptly while processing a request, the request is processed when the driver starts again. You can instruct the driver start or stop processes to process the unprocessed events if an exception occurred when it was processing the unprocessed events by using the **Allow driver to start if reading unprocessed events fails** parameter. If you want to remove an unprocessed request, check the request in the DriverStorage attribute and then manually remove the request. By default, the driver storage can accommodate 500 requests. NetIQ recommends you to use a value less than the default value. Using a higher value causes memory build up issue. You can control this behavior by using the **Maximum number of command's in the driver storage** parameter.

Perform the following activities to set up a Role and Resource Service driver for multi-threaded services:

1. Define a unique data set in Mapping Table. For more information, see “Modifying the Default Mapping Table Object” on page 145.
2. Enable the driver for multi-threaded services using the driver configuration parameters. For more information, see “Configuring the Driver” on page 145.
3. Deploy the driver to the Identity Vault. For more information, see “Deploying the Driver” on page 146.

Prerequisites

- Identity Manager 4.7
- Designer 4.7

Defining a Unique Data Set

A unique data set comprises of users, group, or sub-containers that do not overlap. You can define a data set by using the default mapping table provided with the driver. The mapping table has two columns. The first column lists the criteria which defines the unique data set and the second column has the key for that unique data set.

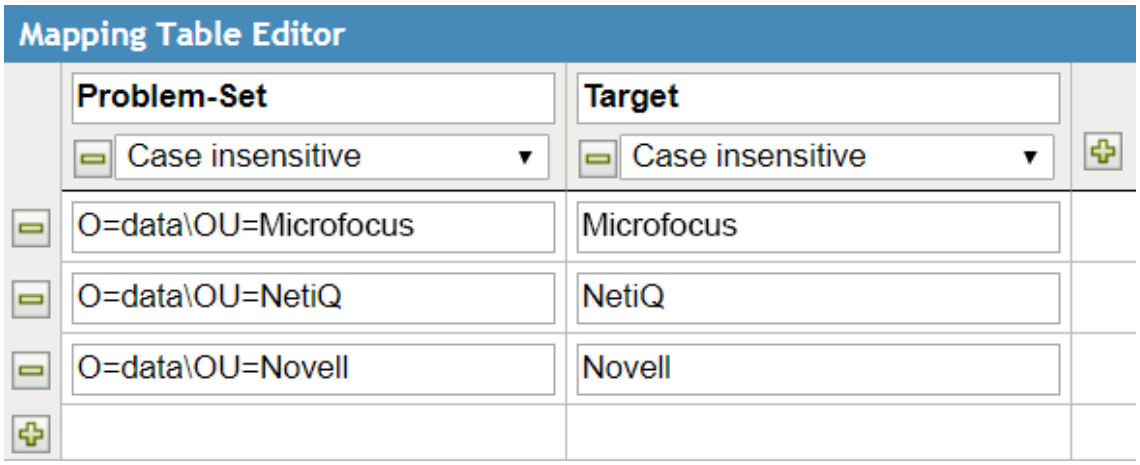
To use a different criterion, you must define a unique data set in a mapping table. For example, you can specify the container DN as a criterion for identifying a data set. You can either create a policy or select an existing policy and add the mapping table to it.

By default, the driver supports creation of unique data sets based on Organizational Units (OU's). To use a different criterion, you must define the data set in the mapping table and modify the policy rules that are linked with the mapping table. For example, if you want to specify `Location` as a criterion for creating a data set, the mapping table must contain location details. You also need to modify the policy rules linked with the mapping table.

Modifying the Default Mapping Table Object

1. In the Outline view, right-click the mapping table object.
2. Select **Open the editor**.
3. In the File Conflict message, click **Yes** to save the project before opening the Mapping Table editor.
4. Specify a column name and data type, and then click **Close**. For example, `Microfocus`.

   Column names must be unique. The data type lets you specify the column values as **Case Sensitive**, **Case Insensitive**, or **Numeric**.
5. Select **New Value** and specify a cell value. For example, **Problem-Set:** `O=data\OU=Microfocus` **Target:** `Microfocus`.
6. Press **Ctrl+S** to save the mapping table object.



Identity Manager automatically links this mapping table object to the following policies:

- Resolve the disjoint set for which the role request or resource request belongs to
- Resolve the disjoint set for which the user resynchronization request belongs to

Configuring the Driver

1. Open your project in Designer.
2. In the Outline pane, expand **Identity Vault.**
3. Right-click the driver set containing your driver and select **Driver > Driver Configuration**.
4. For **Enable multi-threaded Role and Resource driver** option, select **true**.

The following options are displayed:

- **Allow driver to start if reading unprocessed events fails:** Specify whether the driver should start or stop when it encounters an exception while reading unprocessed requests. By default, the value is set to **false**, which disallows the driver to read the unprocessed requests.
- **Maximum number of command's in the driver storage:** Specify the number of commands that the driver storage will store. The default value is **500**. NetIQ recommends to use a value less than the default value to avoid any memory issues.

5 Click **OK**.

6 Continue with "Deploying the Driver" on page 146.

Deploying the Driver

1 Right-click the driver that you configured in "Configuring the Driver" on page 145.

2 Select **Live > Deploy**.

3 Restart the Identity Vault for the changes to take effect.

Limitations

- The driver does not support parallelization of events for the same unique data set. Such events are sequentially processed.
- The driver does not support multiple threads for a specific unique data set. Each unique data set has a dedicated thread.

Troubleshooting

A multi-threaded driver logs messages to the driver log. The messages are appended with Thread IDs. To troubleshoot the driver, check the Thread IDs as a starting point for analysis. For more information, see "Troubleshooting Multi-Threaded Role and Resource Service driver Issues" on page 630.

13 Configuring Identity Applications Clustering and Permission Clustering

You can configure Identity Applications clustering and permission clustering to use TCP or UDP

Configuring Identity Applications Clustering to Use TCP or UDP

You can configure the Identity Applications clustering to use TCP or UDP in local or global settings. The configuration process must be completed on each server in the cluster because the local settings are saved on the file system for each server.

To configure the Identity Applications clustering to use TCP or UDP:

1. Log in to Identity Applications as the User Application Administrator and go to **Configuration > Caching and Cluster**.
2. In the Caching Management page that opens, click **Cluster Cache Configuration**.
3. Navigate to **Local Settings** and perform the following actions:
   - **3a** Enable the **Enable Local** and **Local** check boxes in the **Cluster Enabled** row.
   - **3b** Enable the **Enable Local** check box in the **Cluster Properties** row and specify a value in the text field in the **Local** column.

     To use TCP, specify the following string:

     ```
     TCP(bind_port=$bindport):TCPPING(initial_hosts=$host_details;port_r
     ange=5):MERGE3(min_interval=10000;max_interval=30000):FD_SOCK:FD(ti
     meout=2500;max_tries=5):VERIFY_SUSPECT(timeout=1500):BARRIER:pbcast
     .NAKACK2(use_mcast_xmit=false):UNICAST3:pbcast.STABLE(desired_avg_g
     ossip=50000;max_bytes=4M):pbcast.GMS(print_local_addr=true;
     join_timeout=2000):MFC(max_credits=2M;min_threshold=0.4):FRAG2(frag
     _size=60K):pbcast.STATE_TRANSFER
     ```

     Specify the values for `bind_port` and `initial_hosts`. For example:

     ```
     TCP(bind_port=7815):TCPPING(initial_hosts=192.168.20.10[7815],192.1
     68.20.20[7815];port_range=5):MERGE3(min_interval=10000;max_interval
     =30000):FD_SOCK:FD(timeout=2500;max_tries=5):VERIFY_SUSPECT(timeout
     =1500):BARRIER:pbcast.NAKACK2(use_mcast_xmit=false):UNICAST3:pbcast
     .STABLE(desired_avg_gossip=50000;max_bytes=4M):pbcast.GMS(print_loc
     al_addr=true;
     join_timeout=2000):MFC(max_credits=2M;min_threshold=0.4):FRAG2(frag
     _size=60K):pbcast.STATE_TRANSFER
     ```

     To use an external IP address, specify `bind_addr`. For example:

```
TCP(bind_addr=idmapps;bind_port=$bindport):TCPPING(initial_hosts=$h
ost_details;port_range=5):MERGE3(min_interval=10000;max_interval=30
000):FD_SOCK:FD(timeout=2500;max_tries=5):VERIFY_SUSPECT(timeout=15
00):BARRIER:pbcast.NAKACK2(use_mcast_xmit=false):UNICAST3:pbcast.ST
ABLE(desired_avg_gossip=50000;max_bytes=4M):pbcast.GMS(print_local_
addr=true;
join_timeout=2000):MFC(max_credits=2M;min_threshold=0.4):FRAG2(frag
_size=60K):pbcast.STATE_TRANSFER
```

To use UDP, specify the following string:

```
UDP(mcast_addr=228.8.8.8;mcast_port=45654):PING(timeout=5000):FD(ti
meout=10000;max_tries=5):VERIFY_SUSPECT:pbcast.NAKACK:UNICAST:pbcas
t.STABLE:FRAG:pbcast.GMS:pbcast.FLUSH(timeout=0)
```

The properties in these strings are defined by JBoss. For more information, see the JGroups 4.0.12 documentation.

4. Save the changes. These changes are written to the local file system for your server. Remember to make these changes for all servers in the cluster. Any server that does not have these changes uses the global settings values.
5. Restart the server.

Configuring Permission Clustering to Use TCP or UDP

You can configure the permission clustering to use TCP or UDP. The configuration process must be completed on each server in the cluster because the local settings are saved on the file system for each server.

To configure the permission clustering to use TCP or UDP:

1. Log in to Identity Applications as the User Application Administrator and go to **Configuration > Caching and Cluster**.
2. In the Caching Management page that opens, click **Cluster Cache Configuration**.
3. Navigate to **Global Settings** and perform the following actions:
   - **3a** Enable the **Global** check box in the **Permission Index Cluster Enabled** row.
   - **3b** Click **Permission Index Cluster Properties**, specify a value in the text field in the **Local** column:

     To use TCP, specify the following string:

     ```
     TCP(bind_port=$bindport):TCPPING(initial_hosts=$host_details;port_r
     ange=5):MERGE3(min_interval=10000;max_interval=30000):FD_SOCK:FD(ti
     meout=2500;max_tries=5):VERIFY_SUSPECT(timeout=1500):BARRIER:pbcast
     .NAKACK2(use_mcast_xmit=false):UNICAST3:pbcast.STABLE(desired_avg_g
     ossip=50000;max_bytes=4M):pbcast.GMS(print_local_addr=true;
     join_timeout=2000):MFC(max_credits=2M;min_threshold=0.4):FRAG2(frag
     _size=60K):pbcast.STATE_TRANSFER
     ```

     Specify the values for `bind_port` and `initial_hosts`. For example:

```
TCP(bind_port=7815):TCPPING(initial_hosts=192.168.162.10[7815],192.
168.162.11[7815];port_range=5):MERGE3(min_interval=10000;max_interv
al=30000):FD_SOCK:FD(timeout=2500;max_tries=5):VERIFY_SUSPECT(timeo
ut=1500):BARRIER:pbcast.NAKACK2(use_mcast_xmit=false):UNICAST3:pbca
st.STABLE(desired_avg_gossip=50000;max_bytes=4M):pbcast.GMS(print_l
ocal_addr=true;
join_timeout=2000):MFC(max_credits=2M;min_threshold=0.4):FRAG2(frag
_size=60K):pbcast.STATE_TRANSFER
```

For more information, see the JGroups 4.0.12 documentation.

To use UDP, specify the following string:

```
UDP(mcast_addr=228.8.8.8;mcast_port=45654):PING(timeout=5000):FD(ti
meout=10000;max_tries=5):VERIFY_SUSPECT:pbcast.NAKACK:UNICAST:pbcas
t.STABLE:FRAG:pbcast.GMS
```

The properties in these strings are defined by JBoss. For more information, see the JGroups 4.0.12 documentation.

4 (Conditional) Perform the following actions for using TCP:

4a Set `start_port`. This value must take into account the ports that are already in use and the value for `port_range` to avoid port conflicts. Depending on your configuration, you may need to troubleshoot to find an unused port.

4b Change the IP addresses to include the IP addresses of all the nodes in the cluster and their `start_port` values. The list should begin with the local IP address.

5 Save the changes. These changes are written to the local file system for your server. Remember to make these changes for all servers in the cluster. Any server that does not have these changes uses the **Global Settings** values.

6 Restart the server.

III Identity Applications Administration

This section provides information about managing the roles and resources that you intend to grant to users in your organization.

- Chapter 14, “Creating and Managing Roles,” on page 153
- Chapter 15, “Creating and Managing Resources,” on page 163
- Chapter 16, “Creating and Managing Delegations,” on page 173
- Chapter 17, “Separation of Duties Constraints,” on page 175
- Chapter 18, “Using Controlled Permission Reconciliation Services,” on page 177
- Chapter 19, “Configuring Identity Applications Default Settings,” on page 189
- Chapter 20, “Configuring Email-Based Approval,” on page 211
- Chapter 21, “Configuring and Managing Objects for Entities,” on page 213

14 Creating and Managing Roles

A **role** defines a set of permissions related to one or more target systems or applications. For example, a user administrator role might be authorized to reset a user's password, while a system administrator role might have the ability to assign a user to a specific server.

Identity applications allow you to create or modify roles, associate resources and roles to a role, and assign a role to users.

Go to **Administration > Roles** to create and manage roles. This page displays the list of roles in your organization. For more information, click ⓘ on the Dashboard.

To create and manage roles, you must have one of the following identity applications roles:

- Role Administrator
- Role Manager

To modify the default role settings, see "Configuring Default Roles Settings" on page 189.

NOTE: You must have Security Administrator role to modify the system roles.

You can perform the following operations on the roles within your organization:

- "Listing Roles" on page 153
- "Creating a New Role" on page 154
- "Editing Roles" on page 154
- "Managing the Role and Resource Service Driver" on page 159

Listing Roles

By default, all roles are listed on the **Roles** page. If you want to change the default settings, go to **Your ID** > **Settings** > **Customization** > **General** and select the **Enable Eager Search Results in Roles and Resources Page** as required.

The **Roles** page displays all the roles within your organization. You can search roles using role name or description. You can also filter roles based on role level and categories. If you wish to see the columns other than default columns, you can customize the columns.

IMPORTANT: Do not use the following special characters in the search bar to find a role: `< > , ; \ " + # = / | & ' ! @ $ %`. Additionally, the characters that you have defined in the `com.novell.xss.blacklist.workflow` property of the `ism-configuration.properties` file are also not supported.

For more information, click ⓘ on the dashboard.

Creating a New Role

To create a new role, click the + icon and specify the fields marked with an asterisk (*).

You cannot change the specified role level and subcontainer information later. If you want to have different role level and subcontainer information, create a new role with the required information.

IMPORTANT

- Do not use the following special characters in the **ID** field: `< > , ; \ " + # = / | & * ' ! @ $ %`
- Do not use the following special characters in the **Name** field: `< > , ; \ " * ~ + # = / |` `&`

Additionally, the characters that you have defined in the `com.novell.xss.blacklist.workflow` property of the `ism-configuration.properties` file are also not supported.

For more information, click ⓘ on the dashboard.

Editing Roles

You can modify all the role parameters except **Level** and **Subcontainer**. Identity Manager Dashboard allows you to edit each role separately or multiple roles at once.

Editing individual roles: To edit an individual role, select a role from the list that you want to edit. You can perform the following operations:

- Changing the role details such as role name, description, and categories. Adding or removing role owners.

  NOTE: The role owner can be a user, a group, or a container. The role owner does not automatically have the authorization to administer changes to a role definition. In some cases, the owner must ask a Role Administrator to perform any administration actions on the role.

- Modifying the role approval and revocation process, see “Changing Approval and Revocation Process” on page 156.
- Associating resources within your organization to the selected role. See, “Mapping Resources to Roles” on page 156.
- Assigning the selected role to the required users in your organization. See,“Assigning Roles to Users” on page 157.
- Checking the request status of the users requesting for the selected role.
- Mapping other roles to the selected roles. See, “Mapping Roles to Roles” on page 158.

Editing multiple roles at once: You can edit multiple roles as a group instead of requiring you to repeat those actions on each role individually. Select the roles you want to manage from the list of roles. You can change **Categories**, **Owners**, and **Approval Details** for the roles you selected. Also, you

can **Append** or **Overwrite** values for **Categories** and **Owners** for the selected roles. Append option allows you to add values without altering the existing entries. Overwrite option replaces the values that are entered for the existing values.

For more information, see "Changing Approval and Revocation Process" on page 156.

Delete Roles: To delete any role from the list, select the role and click **Delete**.

When you instruct the User Application to delete a role, it first sets the role status to Pending Delete. The Role and Resource Service driver then notes the change of status and performs these steps:

- Removes the resource assignments for the role
- Deletes the role itself

The Role and Resource Service driver optimizes this process. However, the process may take some time, depending on the number of users assigned to the role, because the Role and Resource driver must ensure that it does not remove a resource from a user if they have this resource by other means.

For example, to check the status of a Permission role, perform the following steps in iManager:

1. Delete the role in User Application, for example, Email.
2. Log in to iManager.
3. In **Objects** tab, browse to **Driver Set > User Application Driver > AppConfig > RoleConfig > RoleDefs > Level30**.
4. Select the role that you have deleted in Step 1.
5. Click the **nrfStatus** attribute.

   The value of the status is set to 15. The value 15 denotes that the role is in Pending Delete state.

When a role has the status of Pending Delete, you are unable to edit, delete, or assign the role.

What happens to existing role assignments If you delete a role that has an associated resource as well as one or more identities assigned to it, the system removes the resource assignment from each identity that has the associated resource.

NOTE: If you delete a role that has a resource assigned to it (or remove a user from the role), the system removes resource assignments for users in that role, even if those resources were first assigned directly. The reason for this is that the system assumes that the last authoritative source for a resource assignment is the controller of that resource, as illustrated by the following scenario:

1. A resource is created with an entitlement.
2. A user is assigned to the resource created above.
3. A role is created that is bound to the resource created in the first step above.
4. The same user is then assigned to the role created above.
5. The user is removed from the role.

In this situation, the user gets removed from the resource even though they had the resource assigned directly. Initially, the resource assignment is considered the authoritative source. However, when the user is assigned to a role that is associated with the same resource, the role becomes the authoritative source.

Changing Approval and Revocation Process

After you create a role, you can modify it to define the approval process for that role. An approver can be a user, group, container, or a specific role.

You can define the approval process for a role using one of the following options:

- **Serial Approval:** Specify multiple approvers, and reorder the selected approvers to define the approval hierarchy.
- **Quorum Approval:** Specify the approvers, then use the slide bar to specify the percent of those approvers that are required to grant access.
- **Custom:** Specify the customized approval process from the list that you want to use. The list displays the workflows that are defined using Designer.

NOTE: You must set up this approval process in Designer. For more information, see *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.

If you choose **None**, no approvers are required for the role.

You can choose to have a revoke process or not. If **Revoke Process Required** is enabled, the revocation process follows the same process that is defined for role approval.

Mapping Resources to Roles

A role defines a set of duties for an individual, to carry the duties for the assigned role might require certain resources. For example, a Facilities Manager role should also have access to the Office printer. In this case, you can map the Office printer (resource) to the Facilities Manager role. When anyone requests for the Facilities Manager role, granting permission to the Facilities Manager role also grants an access to the Office printer.

While editing a role you can map resources to the selected role. **Roles** page also allows you to map resources to the role with more assistance such as:

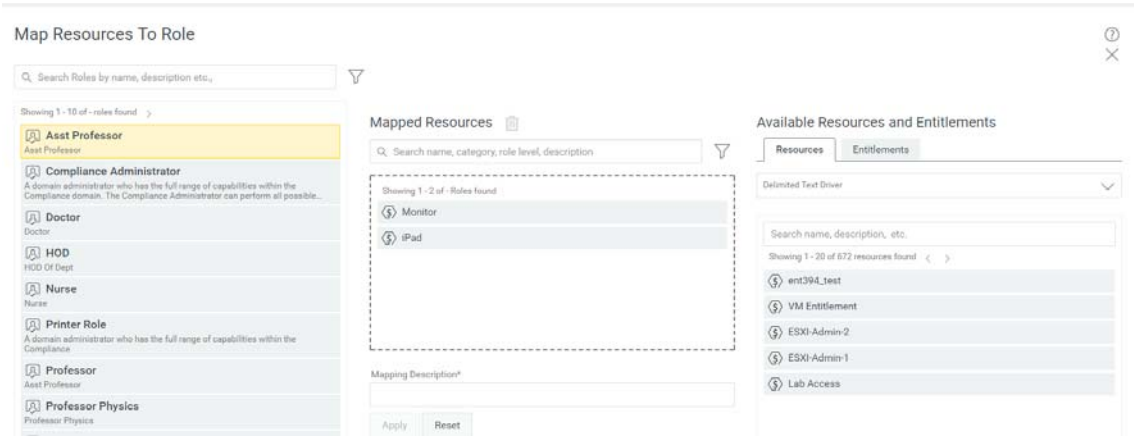
- View the existing resource to role mappings at a time.
- Allows you to map resources/entitlements to role directly.
- Search and filter roles, resources, and entitlements for mapping on the same page.

Click the  icon and map the resources to the required roles and perform the following steps:

1. Select the role from the roles list.
2. Drag and drop the resources/entitlements that you want to map from the **Available Resources and Entitlements** list to **Mapped Resources**.
3. Specify the **Mapping Description**.
4. Click **Apply**.

The following image is an example for mapping resources to a role:

Figure 14-1 *Map Resources to Role*



Search Roles

Shows the list of roles in the organization.

Mapped Resources

Shows the list of resources mapped to the selected role.

Available Resources and Entitlements

Shows the list of available resources and entitlements in the organization.

Assigning Roles to Users

You can directly assign the selected role to any user, group, and container in your organization. While assigning roles, you can set the effective date and expiration date for this assignment.

1. Select a role that you want to assign to users.
2. In **Role Assignments**, Click **+**.
3. Specify the **Initial Request Description**, and mention the **Recipients** from the list.

   NOTE: In **Initial Request Description**, describe the purpose of assigning a role to the mentioned users in **Recipients** list.

   In **Recipients**, you can mention users, group, and container from the list.

   ---

4. (Conditional) Set the **Effective Date** and **Expiration Date** for this assignment.

   If you do not set effective and expiration date, the effective date will be set to the present day and no expiry for this assignment.
5. Click **Assign Role**.

For more information, click ⓘ on the dashboard.

Mapping Roles to Roles

Role levels define role hierarchy. The roles hierarchy supports three levels. Roles defined at the highest level (called Business Roles) define operations that have business meaning within the organization. Mid-level roles (called IT Roles) supports technology functions. Roles defined at the lowest level of the hierarchy (called Permission Roles) define lower-level privileges.

A higher-level role automatically includes privileges from the lower-level roles that it contains. For example, a Business Role automatically includes privileges from the IT Roles that it contains. Similarly, an IT Role automatically includes privileges from the Permission Roles that it contains.

Role relationships are not permitted between peer roles within the hierarchy. In addition, lower-level roles cannot contain higher-level roles.

You can modify the label used for each role level in the User Application by defining localized strings for the level's **Name** and **Description** in the role configuration editor.

You can define Parent Roles and Child Roles for the selected role.

Parent Roles

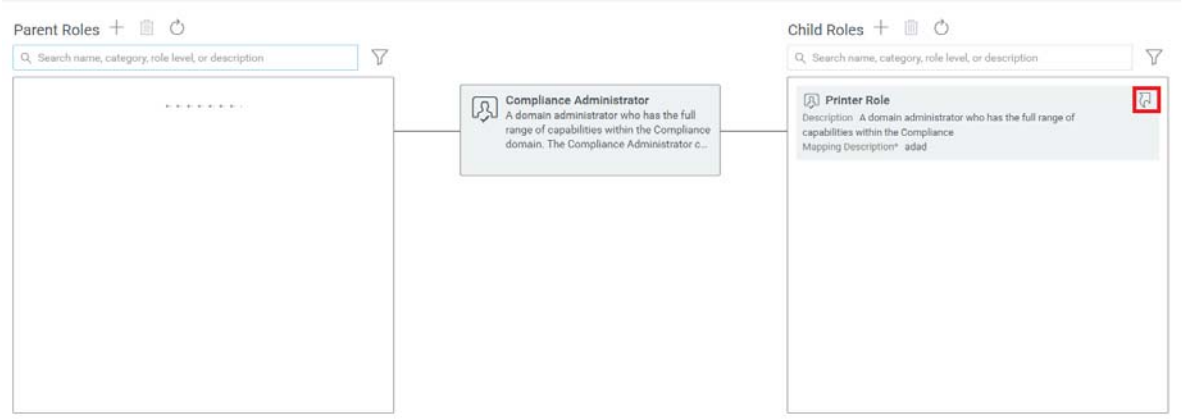
Roles which are higher to the selected role. These roles have all the permissions of the selected role in addition to the permissions specified for these roles.

Child Roles

Roles which are lower to the selected role. The selected role has all the permissions of the child roles in addition to the permissions specified for the selected role.

A child role must have a lower role level than the parent role, and the parent role is automatically assigned the privileges assigned to the lower-level roles.

Figure 14-2 *Map Roles to Roles*



TIP: To see the role relationship of a Parent or Child Role, click on the role that you wish to see the hierarchy.

Managing the Role and Resource Service Driver

On occasion, you might want to change the settings for the Role and Resource Service driver or update the indexes that it uses to display roles in the identity applications.

Configuring the Role and Resource Service Driver Settings

After creating the Role and Resource Service driver at installation time, you can optionally modify some of the driver configuration settings in iManager.

1. In iManager, click **Identity Manager> Identity Manager Overview**.
2. Browse to the driver set where the driver exists, then click **Search**.
3. Click the upper-right corner of the Role and Resource Service driver icon, then click **Edit Properties**.
4. Click on the **Driver Configuration** tab.
5. Scroll down to the **Driver Settings** section of the page.
6. Make any changes you would like to the settings, and click **OK** to commit your changes.

You can modify the following standard driver settings (listed under **User Application/Workflow Connection** on the Driver Configuration page), which get their initial values at installation time:

Table 14-1 *Standard Driver Settings*

Option	Description
User Application Driver DN	The distinguished name of the User Application driver object that is hosting the role system. Use the eDirectory format, such as UserApplication.driverset.org, or browse to find the driver object. This is a required field.
User Application URL	The URL used to connect to the User Application in order to start Approval Workflows. This is a required field.
User Application Identity	The distinguished name of the object used to authenticate to the User Application in order to start Approval Workflows. This needs to a user who has been assigned as a Provisioning Administrator for the User Application. Use the eDirectory format, such as admin.department.org, or browse to find the user. The identity needs to be entered in LDAP format (for example, cn=admin,ou=department,o=org), rather than dot format. Note that this is different from the format required at driver install time, where dot notation is expected. This is a required field.

Option	Description
User Application Password	Password of the account specified in the User Application Identity field. The password is used to authenticate to the User Application in order to start approval workflows. This is a required field.
Reenter User Application Password	Re-enter the password of the account specified in the User Application Identity field.

In addition, you can modify the following additional settings (listed under **Miscellaneous** on the Driver Configuration page) to customize the behavior of the Role and Resource Service driver:

Table 14-2 *Additional Settings for Customizing the Role and Resource Service Driver*

Option	Description
Number of days before processing removed request objects	Specifies the number of days the driver should wait before cleaning up request objects that have finished processing. This value determines how long you are able to track the status of requests that have been fulfilled.
Frequency of reevaluation of dynamic and nested groups (in minutes)	Specifies the number of minutes the driver should wait before reevaluating dynamic and nested groups. This value determines the timeliness of updates to dynamic and nested groups used by the User Application. In addition, this value can have an impact on performance. Therefore, before specifying a value for this option, you need to weigh the performance cost against the benefit of having up-to-date information in the User Application.
Generate audit events	Determines whether audit events are generated by the driver. For details on audit configuration, see Chapter 8, “Setting Up Logging in the Identity Applications,” on page 67.

Indexing for the Role and Resource Service Driver

The Role and Resource Service driver has relevant indexes in the Identity Vault for roles definitions. If you upload a large number of roles, the indexing of these values may take some time. You can monitor these indexes under Index Management in iManager.

Here is the list of Index Names for the indexes for the Role and Resource Service driver:

```
nrf(Object Class)
nrf(nrfMemberOf)
nrf(nrfStatus)
nrf(nrfStartDate)
nrf(nrfNextExpiration)
nrf(nrfParentRoles)
nrf(nrfChildRoles)
nrf(nrfCategory)
nrf(nrfRoleCategoryKey)
nrf(nrfLocalizedNames)
nrf(nrfLocalizedDescrs)
nrf(nrfRole)
nrf(nrfResource)
```

15 Creating and Managing Resources

A resource is any digital entity such as a user account, computer, or database that a business user needs to be able to access. Creating a resource with an entitlement helps you manage the entitlements in Identity Applications. For more information, see Resources in "Providing Permissions to Users" on page 36.

Identity applications allow you to create and manage resources with or without entitlements.

Go to **Administration > Resources** to create and manage resources. This page displays the list of resources in your organization. For more information, click ⓘ on the Dashboard.

To create and manage resources, you must have one of the following identity applications roles:

- Resource Administrator
- Resource Manager

To view the default resource settings, see "Configuring Default Resource Settings" on page 190.

You can perform the following operations on the resources within your organization:

- "Listing Resources" on page 163
- "Creating a New Resource" on page 164
- "Editing Resources" on page 165
- "Enabling Drivers for Resource Mappings" on page 168
- "Creating a List to Improve Resource Request Forms" on page 169
- "Resource Assignments" on page 170

Listing Resources

By default, all resources are listed on the **Resources** page. If you want to change the default settings, go to **Your ID** > **Settings** > **Customization** > **General** and select the **Enable Eager Search Results in Roles and Resources Page** as required.

The **Resources** page lists all the resources alphabetically. You can search for resources using resource name or description. You can also filter resources based on resource level and categories.

IMPORTANT: Do not use the following special characters or a whitespace in the search bar to find a resource: `< > , ; \ " + # = / | & ' ! @ $ %`. Additionally, the characters that you have defined in the `com.novell.xss.blacklist.workflow` property of the `ism-configuration.properties` file are also not supported.

For more information, click ⓘ on the Dashboard.

Creating a New Resource

To create a new resource, click the + icon. You can create a resource with or without entitlement.

With Entitlement

If you choose to create a resource with this option, select the driver or entitlement for which you want to create a resource. The **Resource Name** and **Resource Description** fields are auto-populated based on the selected driver or entitlement.

IMPORTANT: You must rename the **Resource Name** field to a valid name if it contains any of these `[ < > , ; \ " + # = / | & * ' ! @ $ % ]` special characters or a whitespace. Additionally, the characters that you have defined in the `com.novell.xss.blacklist.workflow` property of the `ism-configuration.properties` file are also not supported.

You can choose to tag an entitlement value during resource creation or allow the user to select entitlement values at the time of the request.

- **Tag an entitlement value to a resource:** Specify the necessary entitlement values for the selected driver or entitlement. For every specified entitlement values, a separate resource is created.

  A user can request this resource which has the defined entitlement value. For example, you created a resource by selecting a printer as an entitlement value for the Office Resources entitlement. A user can request for Office Resources entitlement that has the defined Printer entitlement value.

- **Allow users to select entitlement values at the time of request:** Select **Map Entitlement Values at Resource Request time** and specify **Label for Value field**.

  For this type of resources, a user can select the required entitlement values from the list, while requesting for this resource. For example, you created a resource for Office Resources entitlement without defining an entitlement value, a user can select any entitlement value from the list for the Office Resources entitlement at the time of the request.

  To enable this option for the logical systems within the connected system, you must create a separate resource for each logical systems.

  NOTE: Select **Allow this resource and entitlement to be assigned multiple times with different values** only if you want to allow users to request this resource multiple times with different values.

Without Entitlement

If you choose to create a resource with this option, specify the fields marked with an asterisk (*).

You cannot change the specified resource **Level** and **Subcontainer** information later. To change this information, you must delete this resource and recreate the resource with the required resource level and subcontainer information.

IMPORTANT

- Do not use the following special characters or a whitespace in the **ID** field: `< > , ; \ " + # = / | & * ' ! @ $ %`.
- Do not use the following special characters or a whitespace in the **Name** field: `[ < > , ; \ " + # = / | & * ' ! @ $ % ]`

Additionally, the characters that you have defined in the `com.novell.xss.blacklist.workflow` property of the `ism-configuration.properties` file are also not supported.

You can set the expiration period for the resources. The permission to the resource will be revoked from the user, once it crosses the specified expiration period.

For more information, see "Setting Expiration Period for the Resource" on page 166.

Editing Resources

You can modify all the resource parameters except **Level** and **Subcontainer**. Identity applications allow you to edit each resource separately or multiple resources at once.

Editing individual resources: Select a resource from the list that you want to edit and perform any of the following operations:

- Changing the resource details such as resource name, description, and categories. Adding or removing resource owners.
- Set expiration period for the selected resource. See "Setting Expiration Period for the Resource" on page 166,
- Modifying the resource approval and revocation process. See, "Changing the Approval or Revocation Process" on page 166.
- Modifying a resource to add entitlements or allowing that resource and entitlement to be assigned multiple times with different values. For more information, see "With Entitlement" on page 164 section.

  NOTE: If a resource is already assigned to user, you cannot modify the entitlements.

- Assigning the selected resource to the required users in your organization. See, "Assigning Resource to Users" on page 167.
- Checking the request status of the users requesting for the selected resource.
- Update the resource form for the selected resource. See, "Updating the Resource Request Form" on page 167.

Editing multiple resources at once: You can edit multiple resources as a group instead of requiring you to repeat those actions on each resource individually. Select the resources you want to manage from the list of resources. You can change **Categories**, **Owners**, and **Approval Details** for the resources you selected. Also, you can **Append** or **Overwrite** values for **Categories** and **Owners** for the selected resources. Append option allows you to add values without altering the existing entries. Overwrite option replaces the values that are entered for the existing values.

For more information, see "Changing the Approval or Revocation Process" on page 166.

Delete Resources: To delete any resource from the list, select the resource and click **Delete**.

What happens to existing resource assignments When you a delete a resource that already has one or more identities assigned to it, the system removes the resource from those identities. If the resource has been associated with a role, the system also removes all role associations that pertain to the deleted resource.

Setting Expiration Period for the Resource

To set the expiration period, enable **Expiration required** and set the number of **Days**/**Months**/**Years** when the access to the selected resource(s) should expire.

Expiration period sets the expiration date for a resource from the date of assignment.

Users can also request for resources in **Access > Request** page, for a specific period. For more information, see Requesting Permissions in *NetIQ Identity Manager - User's Guide to the Identity Applications*.

For more information, click ⓘ on the Dashboard.

Changing the Approval or Revocation Process

After you create a resource, you can modify the resource information and define the approval process for it. You can choose the role approval process to override the resource approval process.

You can define the approval process for a resource using one of the following options:

- **Serial Approval:** Specify multiple approvers, and define the order by selecting an approver and moving that approver earlier or later in the order by clicking the arrows at the right of the approval list.
- **Quorum Approval:** Specify the approvers, then use the slide bar to specify the percent of those approvers that are required to grant access.
- **Custom:** Specify the customized approval process from the list that you want to use. The list displays the workflows that are defined using Designer.

  NOTE: You must set up this approval process in Identity Manager Designer. For more information, see *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.

If you choose **None**, no approvers are required for assigning the resources.

You can choose to have a revoke process or not. The revocation process can match the approval process. Also, you can define a different revocation process.

You can modify the expiration period for the selected resources. See, "Setting Expiration Period for the Resource" on page 166.

Assigning Resource to Users

You can directly assign the selected resource to any user in your organization.

1. Select a resource that you want to assign to users.
2. In **Resource Assignments**, click **+**.
3. Specify the **Initial Request Description**, and mention the **Recipients** from the list.

   NOTE: In **Initial Request Description**, describe the purpose of assigning a resource to the mentioned users in **Recipients** list.

4. Click **Assign Resource**.

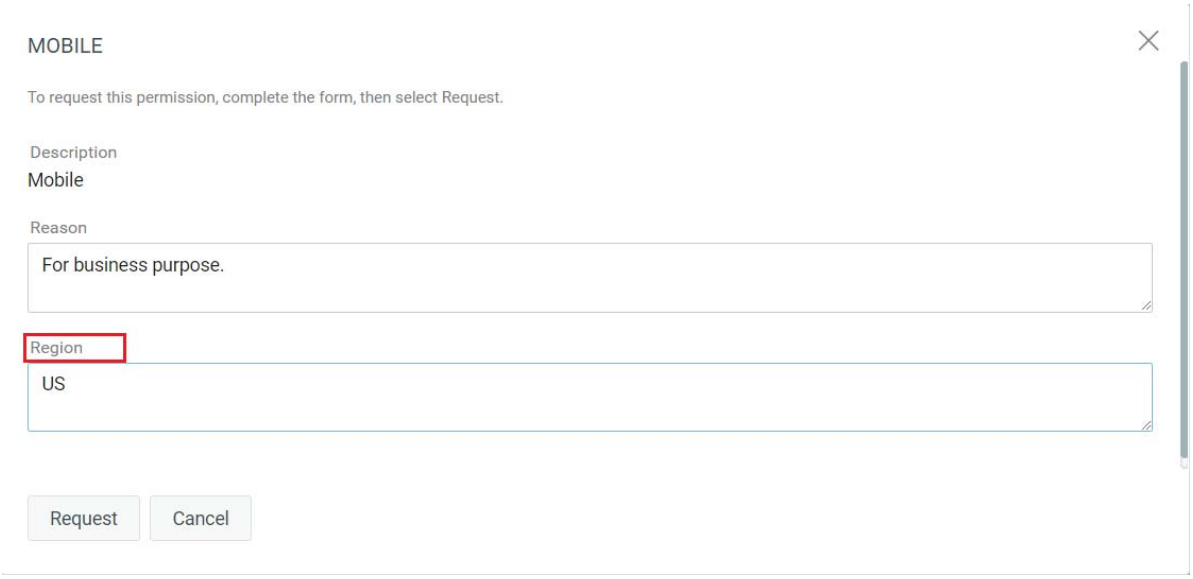
For more information, click ⓘ on the Dashboard.

Updating the Resource Request Form

A resource form is used to gather necessary data to properly assign a resource. Create and define the fields for the resource.

This is an example for resource form. The **Region** field is added for the **Mobile** resource. Following illustration displays the form at the time of the request.

Figure 15-1 *Example Resource Request Form*



You can assign a value to the field and select **At request/assign time** to allow users to specify the values at the time of request or resource assignment. If you want to assign values now, select **Now**.

In **Data Value List**, you can select one of the following data types to add a field into the request form:

Data Type	Description
Integer	This allows you to gather only numerical information about the selected resource. For example, If you require to gather information related to quantity or number of days, weeks, or hours, you can use this data type.
Boolean	This allows you to gather true/false sort of information. For example, If the selected resource is printer, you might need to confirm whether they require a color printer or not. In this case, you can use this data type to provide an option to requesters.
List	This allows you to select values from the list. If you want users to select the defined values, you can provision the options using this data type. For example, If you want to know the time zone of the requester, you can provision this field by listing all the timezones for this field. For more information about creating such lists, see "Creating a List to Improve Resource Request Forms" on page 169.
String	This allows you to gather more information on a resource request. For example, If you want to know the reason for this request or assignment, you can use this data type for a field to gather this information.

NOTE: When you select **Now** option in **Assign Value**, these fields appear on the request form at the time of request or assignment by default. If you want to hide these fields, select **Hide**.

For more information, click ⓘ on the Dashboard.

Enabling Drivers for Resource Mappings

The identity applications includes updated configuration files for the following drivers:

- Active Directory
- GroupWise
- LDAP
- Notes
- eDirectory

- SAP User Management
- SAP GRC Access Control

All of these updated driver configuration files contain a new section on the driver's Global Configuration Values (GCV) page labeled **Role and Resource Mapping**.

To display the configuration options available in the new section, select **show** for the **Show role and resource mapping configuration** GCV.

To enable resource mapping for the driver, select **Yes** for the **Enable resource mapping** GCV.

Depending on the driver's capabilities, one or more lower-level options are displayed once resource mapping is turned on. The Active Directory driver, for example, has three lower-level options:

- **Allow mapping of user accounts**
- **Allow mapping of groups**
- **Allow mapping of Exchange mailboxes**

Each option can be turned on or off individually by selecting **Yes** or **No**.

After saving the changes and restarting the driver, RBPM will detect the driver as enabled for resource mapping.

NOTE: Before Identity Applications detect the driver, it must query the entitlement system. Identity Applications sends the query to the entitlement system every 1440 minutes by default, but you can force the application to send the query immediately using the User Application.

To force the query to run immediately, click **Refresh** in **Entitlement Query Settings**. See, "Configuring Entitlement Query Settings" on page 190.

Creating a List to Improve Resource Request Forms

You can use lists in request forms to display various options for specifying a resource assignment. This section provides instructions for adding lists to the database by executing a few SQL statements. Once these lists have been created, they can be displayed on a request form on the Roles and Resources tab.

The following example shows how you would create a simple set of values for a list:

```
INSERT INTO PROVISIONING_CODE_MAP SET VIEWID='Factory-Locations',
VERSIONNO=1,
DESCRIPTION='Factory Locations', NAME='Factory
Locations',ENTITYKEY='Factory-Locations', ENTITYTYPE=1,
LASTREFRESHED=UNIX_TIMESTAMP();

INSERT INTO PROVISIONING_VIEW_VALUE SET VALUEID='Factory-Locations-1',
VERSIONNO=1, VIEWID='Factory-Locations', PARAMVALUE='Cambridge, MA 02440';

INSERT INTO PROVISIONING_VIEW_VALUE SET VALUEID='Factory-Locations-2',
VERSIONNO=1, VIEWID='Factory-Locations', PARAMVALUE='Provo, UT 97288';
```

The following example uses SQL statements that work with PostgreSQL:

```
INSERT INTO PROVISIONING_CODE_MAP
(VIEWID,VERSIONNO,DESCRIPTION,NAME,ENTITYKEY,ENTITYTYPE,LASTREFRESHED)
VALUES ('Factory-Locations',1,'Factory Locations','Factory-
Locations','Factory-Locations',1,extract(epoch FROM now()));

INSERT INTO PROVISIONING_VIEW_VALUE (VALUEID,VERSIONNO,VIEWID,PARAMVALUE)
VALUES ('Factory-Locations-1','1','Factory-Locations','Cambridge, MA
02440');

INSERT INTO PROVISIONING_VIEW_VALUE (VALUEID,VERSIONNO,VIEWID,PARAMVALUE)
VALUES ('Factory-Locations-2','1','Factory-Locations','Waltham, MA
02451');

INSERT INTO PROVISIONING_VIEW_VALUE (VALUEID,VERSIONNO,VIEWID,PARAMVALUE)
VALUES ('Factory-Locations-3','1','Factory-Locations','Provo, UT 97288');
```

The VIEWID is the primary key for the PROVISIONING_CODE_MAP. The ENTITYTYPE value 1 identifies the map type as a list. The VIEWID is the foreign key for the PROVISIONING_VIEW_VALUE relationship to the PROVISIONING_CODE_MAP table. The VALUEID is the primary key for the PROVISIONING_VIEW_VALUE table.

After the Company Location field has been added to the form, you can specify that the company location value should come from the Company Locations list at request time.

After the Factory Location field has been added, you can specify that the factory location value must come from the Factory Locations list at request time.

At request time, the user can then select the company location and factory location values when assigning the resource.

After the resource has been assigned, the Request Status tab for the resource displays the parameter values chosen from the lists for the request form fields.

Resource Assignments

Resources can be assigned to users only. They cannot be assigned to groups or containers. However, if a role is assigned to a group or container, the users in that group or container might automatically be granted access to the resources associated with the role.

A Resource Administrator can configure the approval process for resources. The approval process for a resource might be handled by one of the following:

- a provisioning request definition
- an external system, by setting the status code on the resource request

If a role assignment initiates a request for a resource, it is possible that the request will not be granted, even though the role is provisioned. The most likely reason for this would be that the necessary approvals were not provided.

When a user requests a resource, the request starts a workflow. The workflow coordinates the approvals needed to fulfill the request. Some requests require approval from a single individual; others require approval from several individuals. In some instances, a request can be fulfilled without any approvals.

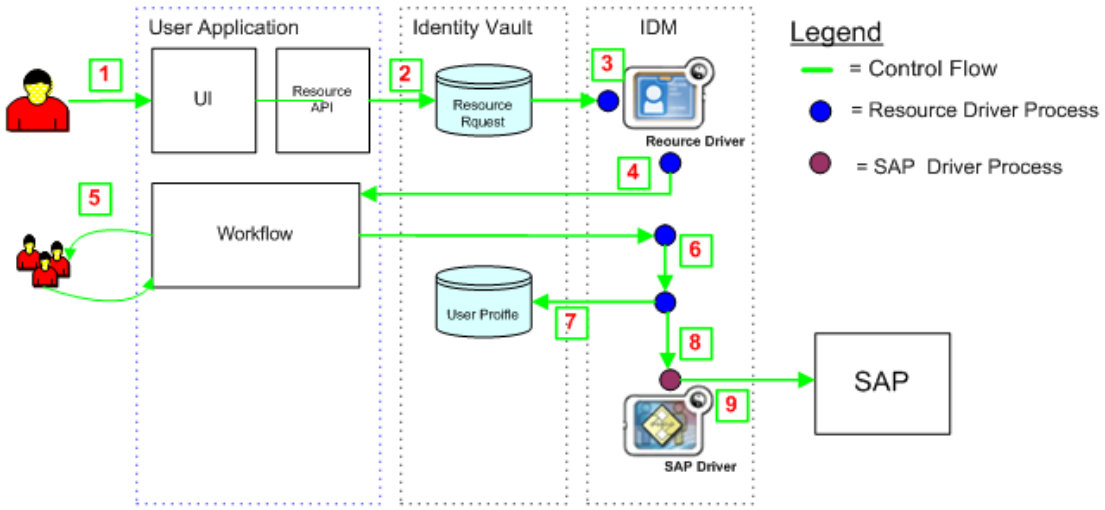
The following business rules govern the behavior of resources within the User Application:

- Resources can only be assigned to a user. The resource can be granted to users in a container or group based on implicit role assignment. But the resource assignment will only be associated with a user.
- Resources can be assigned in any of the following ways:
  - Directly by a user through UI mechanisms
  - Through a provisioning request
  - Through a role request assignment
  - Through a Rest or SOAP interface
- The same resource can be granted to a user multiple times (if this capability has been enabled in the resource definition).
- A resource definition can have no more than one entitlement bound to it.
- A resource definition can have one or more same-entitlement references bound to it. This capability provides support for entitlements where the entitlement parameters represent provisionable accounts or permissions on the connected system.
- Entitlement and decision support parameters can be specified at design time (static) or at request time (dynamic).

Your workflow designer and system administrator are responsible for setting up the User Application for you and the others in your organization. The flow of control for a resource-based workflow, as well as the appearance of forms, can vary depending on how the workflow designer defined the workflow's approval definition in the Designer for Identity Manager. In addition, your job requirements and level of authority determine what you can see and do.

Resource Request Process Flow

The following example shows the process flow for a resource assignment request. In this example, a user requests a resource that grants access to an SAP profile:



1. In the Dashboard, a user requests access to SAP.

2. The Identity Vault creates a User Request object.
3. The Role and Resource Service Driver processes the new request.
4. The Role and Resource Service Driver starts a workflow, and changes the request status.
5. The identity applications perform the approval process. Upon completion of the approval process, the workflow activity changes the request status.
6. The Role and Resource Service driver picks up the change in the status and begins to provision the resource if all of the necessary approvals have been provided.
7. The User Object attributes are updated to include the resource binding and approval information.
8. An entitlement request is made for the SAP Profile.
9. The SAP Driver processes the entitlement and creates the user's profile in SAP.

16 Creating and Managing Delegations

In some organizations, you might be allowed to delegate your tasks to other team members. This feature allows you to delegate your tasks to multiple users based on the selected Provisioning Request Definitions (PRD).

A delegate user is a user to whom one or more specific tasks appropriate to that user's rights can be delegated so that the delegate can work on those specific tasks on behalf of someone else.

To view delegations, go to **People > Delegation**. All users in the organization can see their delegation assignments.

To create or modify delegation, you must have one of the following roles:

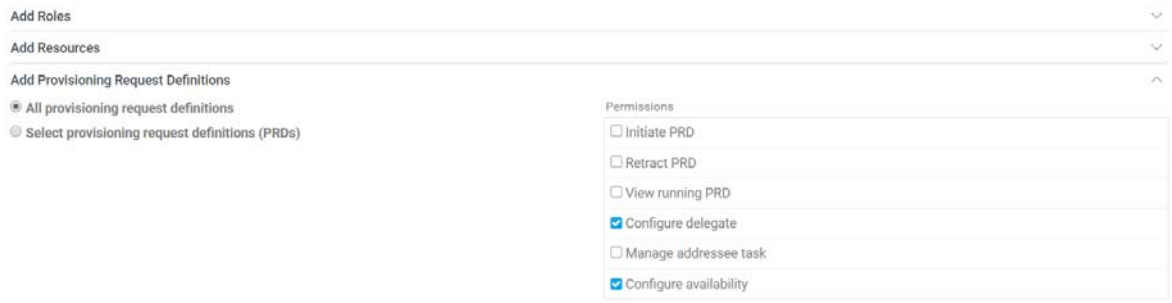
- Provisioning Administrator
- Provisioning Manager
- Team Manager

The Provisioning Administrator and Provisioning Manager have the ability to define delegate assignments for any user in the organization.

The team manager can define delegation assignments for self or for the team members. Make sure that **Configure Delegate** and **Configure Availability** permissions are granted to the team manager for the requested team to create delegation:

Go to **People > Teams** and edit the team permissions to enable delegation feature.

Figure 16-1 *Add Permissions to the Teams to Create Delegation*



NOTE: If team manager wants to create a delegation for self, ensure **Include the selected requesters in the recipients list** is selected.

To modify the default delegation settings, see Section 19, "Configuring Identity Applications Default Settings," on page 189.

For more information about delegations, click ⓘ on the Dashboard.

17 Separation of Duties Constraints

Separation of duties is an important aspect of an organization's security controls because it helps prevent fraud and user error related to user access. In a separation of duties constraint, the conflicting roles must be at the same level in the roles hierarchy.

A Role Administrator can create or modify for roles in the organization.

A SoD constraint represents a rule that makes two roles mutually exclusive unless there is an exception allowed for that constraint. You can define whether exceptions to the constraint are always allowed or are only allowed through an approval flow. When a role assignment results in a potential separation of duties conflict, the initiator has the option to override the separation of duties constraint and provide a justification for making an exception to the constraint.

You can add or delete separation of duties constraints in:

Administration > Separation of Duties page.

To modify the default Separation of Duties settings, see Section 19, "Configuring Identity Applications Default Settings," on page 189.

When a user requests a role that results in a potential SoD conflict, the initiator has the option to override the SoD constraint and provide a justification for making an exception. In some cases, a SoD conflict can cause a workflow to start. The workflow coordinates the approvals needed to allow the SoD exception to take effect.

Your workflow designer and system administrator are responsible for setting up the contents of the **Roles** and **Resources** in the **Administration** tab for you and the others in your organization. The flow of control for a roles-based workflow or SoD workflow, as well as the appearance of forms, can vary depending on how the workflow designer defined the workflow's approval definition in the Designer for Identity Manager. In addition, your job requirements and level of authority determine what you can see and do.

For more information, click ⓘ on the Dashboard.

NOTE: The ability to define custom roles is available only with Identity Manager 4.5 and later.

---

18 Using Controlled Permission Reconciliation Services

Controlled Permission Reconciliation Services (CPRS) helps you to keep the Identity Manager Resource Catalog synchronized with the permissions across connected applications. You can use CPRS for the following activities:

- Initial Permission Onboarding: Allows you to select each entitlement or driver and migrate the permissions of the managed users from the connected application to Resource Catalog.
- Controlled Reconciliation: Select a driver or entitlement and monitor changes to the user permissions in Resource Catalog. You can publish the permission changes to Resource Catalog for specific or all users.

Ensure you have Resource Administrator or User Administrator access to use CPRS.

The following video describes the concepts of CPRS:

 http://www.youtube.com/watch?v=l4-IyRTu2pU

IMPORTANT: CPRS currently supports Active Directory, Multi-Domain Active Directory (MDAD), LDAP, Loopback, Delimited, and REST drivers.

This section provides valuable information for planning a CPRS implementation in your Identity Manager environment.

- "How CPRS Helps" on page 177
- "Prerequisites" on page 178
- "Considerations for Supported Drivers" on page 178
- "Understanding the Components of CPRS" on page 179
- "Managing Permission Reconciliation Settings" on page 181
- "Permission Reconciliation" on page 183
- "Migrating to CPRS" on page 185

How CPRS Helps

Permission Collection and Reconciliation Service (PCRS) helps you create custom entitlements for connected system roles and resources and allows you to synchronize these permission assignment changes to User Applications' Resource Catalog.

When PCRS is configured for an Identity Manager driver, user permissions from that connected application are seamlessly reconciled into Resource Catalog based on PCRS settings configured for the connected application. PCRS does not allow Identity Manager administrators control reconciliation requests. In addition, configuration and troubleshooting of PCRS solution can become

difficult due to involvement of many floating components such as policies, job, mapping tables, and some engine APIs. The primary issues are performance, complexity of implementation and stability/ reliability of the deployments.Therefore, resulting in degraded performance of the system.

CPRS provides an improved solution with the following benefits over PCRS:

- Secures the permission reconciliation model in which reconciliations are controlled by an administrator.
- Requires minimal configuration to get CPRS working after Identity Manager is installed.
- Enhances overall system performance.
- Simpler troubleshooting as all the involved components are integrated with the Identity Applications user interface.

Prerequisites

Before using CPRS, review the following considerations:

- Identity Manager 4.7
- Identity Manager drivers are updated to the latest driver package.

  For more information, see Preparing Drivers to Use CPRS in the *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.
- Identity Manager drivers enabled with entitlements are up and running.

Considerations for Supported Drivers

This section describes all the considerations for the following drivers:

- "MDAD Driver" on page 178
- "Loopback Driver" on page 179
- "REST Driver" on page 179
- "Delimited Text Driver" on page 179

MDAD Driver

MDAD driver allows permission reconciliation for every logical system. You must enable your logical system in Designer or iManager. To enable a logical system in iManager:

1. Login to iManager as an administrator.
2. Right-click the MDAD driver and navigate to **Edit Properties > Global Configuration Values > Entitlement**.
3. Set **Add Logical System information to Entitlement Values** to **Yes** for the entitlement for which you want to use CPRS.

WARNING: This setting (**Yes**) invalidates all the existing resources for the entitlement. This results in loss of all existing resources for the entitlement. Therefore, you need to recreate resources and publish the assignments. For more information, see "Managing Permissions for a MDAD Driver" on page 186.

Loopback Driver

To create associations for the existing users, migrate the user container using **Migrate from Identity Vault** option in iManager. Perform the following tasks to use the existing Loopback driver:

1. Upgrade the driver to the latest Loopback base package.
2. To create associations for the existing users, migrate the user container from Identity Vault to connected application.

REST Driver

Perform the following tasks to use the existing REST driver:

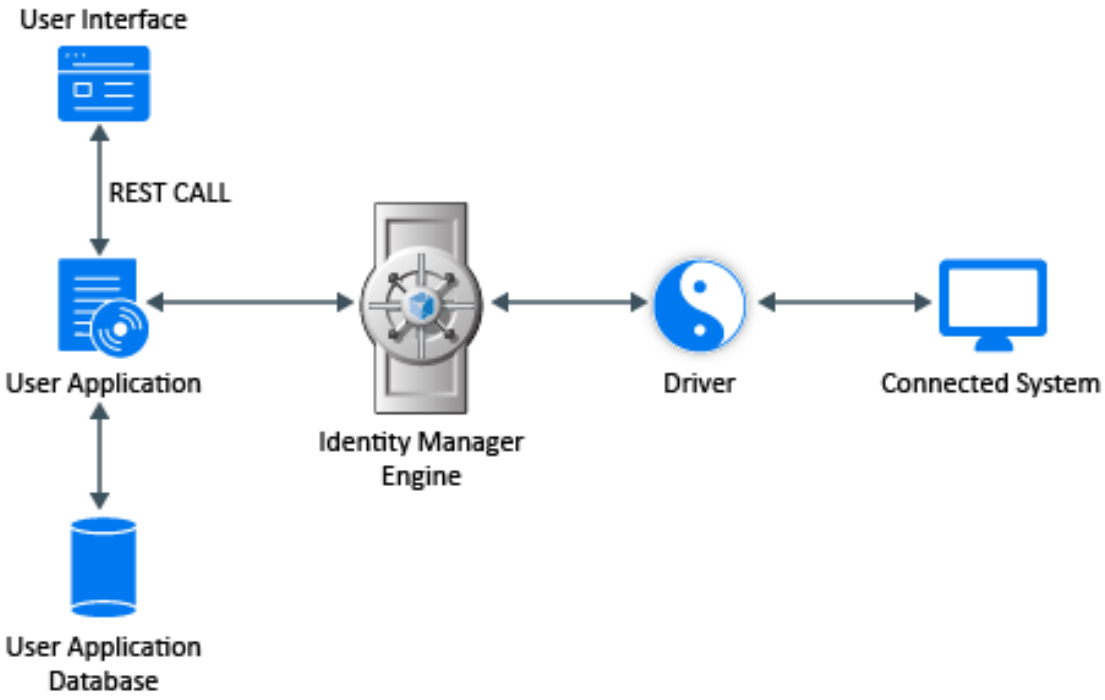
1. Upgrade the Identity Manager to 4.7.1 version.
2. Upgrade the REST driver to 1.0.1.1 version.

Delimited Text Driver

CPRS uses the driver query functionality to fetch permission information from the connected application. Therefore, ensure that the query capability is available in the driver policies for codemap queries and permission queries.

Understanding the Components of CPRS

CPRS is integrated with Identity Manager Dashboard. The following figure depicts different components involved in synchronizing user permissions:



- ***User interface*** - The following web pages are available:
  - **Permission Reconciliation** - Use this page for migrating permissions and to compute the difference in permission assignments between the connected application and Resource Catalog. Use the following icons to manage permission reconciliation, compute assignments, publish assignments, and to view the status of the processes triggered:
    - Manage Permission Reconciliation
    - Compute selected (driver/entitlement) assignments
    - Publish All (driver/entitlement) assignments
    - Status of the process

    For more information, see "Permission Reconciliation" on page 183.
  - **Permission Reconciliation Settings** – Use this page to view the CPRS settings.
  - **Permission reconciliation Settings Edit** – Use this page to modify the permission reconciliation settings for entitlements. For more information, see "Managing Permission Reconciliation Settings" on page 181.
  - **Permission Reconciliation Configuration** - Navigate to **Configuration > Permission Reconciliation**. Use this page to perform the following tasks:
    - Enable permission reconciliation
    - Set the polling time for status checker
    - Set the time for retention for computed permission assignments

    For more information, see "Configuring Permission Reconciliation Settings" on page 191.
- ***User Application Database*** - Contains the computation and the published records.

- ***Identity Manager drivers*** - Used to fetch permissions for data synchronization.
- ***Connected System*** - Any system, directory, database, application, or operating system whose identity information you want to manage.

The following sequence describes how the permission differences between resource catalog and connected systems are computed and published.

1. Log in as a Resource Administrator or User Administrator to **Identity Manager User Interface** > **Administration** > **Permission Reconciliation**.
2. Click the **Manage Permission Reconciliation** icon and click Edit to enable the resources to be used for CPRS.
3. In the **Permission Reconciliation** page, select a driver or entitlement you want to compute or publish.
4. Select an entitlement and click or for a request to be triggered. This action leads to creation of an eDirectory object under a defined container. This object contains information such as, resource to entitlement mapping, operation type, status, and so on.

   The Identity Manager engine is notified when a request is created under the eDirectory container. On detecting a new request, the engine begins to process it. The request object is updated periodically.
5. The difference or change in permission assignments between Resource Catalog and connected applications is called delta. The Identity Manager engine calculates the delta between the Resource Catalog and connected system and stores it in a persistence layer.
6. The Status checker API updates the user application database process records. Once the process status is **Completed** or **Error**, it cleans the request object from the eDirectory container.
7. The User Application REST layer receives the delta of permission assignments from the Identity Manager engine through an LDAP extension.
8. The CPRS computed data is maintained for the configured value in the **Permission Reconciliation Configuration** page > **Retention time for computed permission assignments**.

   For more information, see "Configuring Permission Reconciliation Settings" on page 191.

The following video helps you configure and manage CPRS in Identity Manager Dashboard:

http://www.youtube.com/watch?v=QTh8gnxIVS0

Managing Permission Reconciliation Settings

The **Permission Reconciliation Settings** page allows you to manage the behavior of permission assignment reconciliation between Resource Catalog or Identity Application, and the connected application.

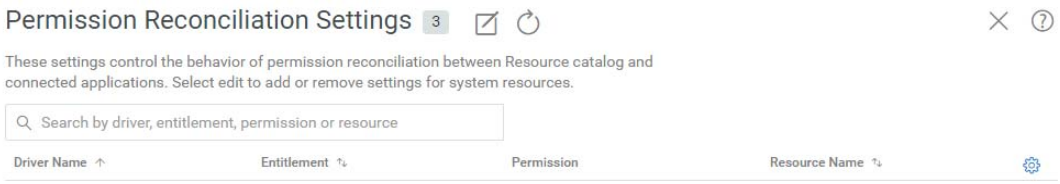
The **Permission Reconciliation Settings** page allows you to configure the resources, to be used by permission reconciliation during delta computation or publish.

Perform the following actions to create settings to reconcile permissions:

Navigate to **Administration** > **Permission Reconciliation**. Click . The **Permission Reconciliation Settings** page appears.

This page lists all the settings made for Identity Applications resources:

- Driver Name
- Entitlement
- Permission
- Resource Name



Ensure a resource is created to configure the permission reconciliation settings.

The existing resources can be mapped in CPRS settings. For more information, click ⓘ on the Dashboard.

NOTE: If settings are listed, click ⚙ to customize the columns. Drag and drop the required columns from **Available Columns** to **Selected Columns**.

Editing Permission Reconciliation Settings

Perform the following steps to edit the Permission Reconciliation settings:

1. Click ✎ to map entitlement from the connected application to the Identity Manager resource.
2. Select an **Entitlement** you want to manage.

   For example: LDAP Driver > User Account Entitlement

   NOTE: If you select MDAD driver, you must select the required **Logical system** to reconcile. By default the first Logical system is selected.

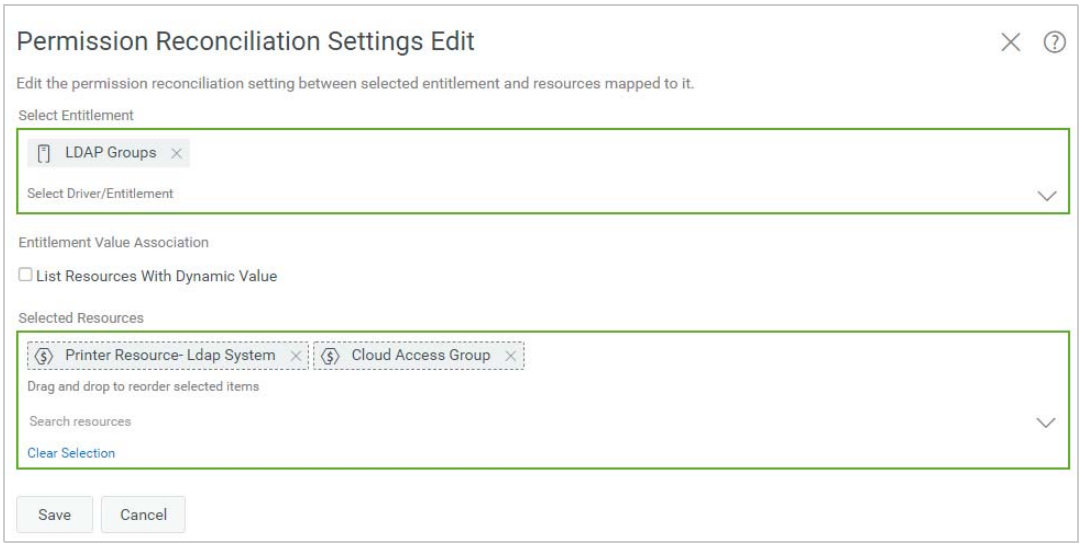
   ---

3. Perform the following actions to create or edit the permission reconciliation settings between the selected entitlement and mapped resources:
   a. (Conditional) In **Entitlement Value Association**, select the **List Resources With Dynamic Value** to list the resources that are not associated with entitlement values. De-select this option to list resources that are associated with entitlement values.

      The list of resources already configured for the selected entitlement is displayed.

   b. Type the resource name you want to select from the list. This lists the resources that are already present in the Resources page.You can select more than one resource for a multivalued entitlement.

      If no resources are listed, you should create a resource with entitlement for the required connected application. To create a new resource, see “Creating a New Resource” on page 164.



4. Click **Save**.

   You can view this setting on the **Permission Reconciliation Settings** page.

Permission Reconciliation

Permission Reconciliation page allows you to compute and publish the permission assignments between Resource Catalog and connected systems.

Ensure that the drivers or entitlements are configured with CPRS settings to compute or publish.

Perform the following actions to publish permissions for the selected driver or entitlement:

1 Navigate to **Administration** > **Permission Reconciliation**.

2 In **Driver or Entitlement**, select a driver or an entitlement that you wish to compute or publish.

IMPORTANT: For a Fan-Out driver (for example MDAD), select a **Logical system**. This option is displayed only for Fan-Out driver. By default the first Logical system is selected.

---

3 Click to compute the difference in assignments between the Resource Catalog and the connected application.

Click to view the process status. You can view the computed assignments data in the **CPRS Assignments** table only when the process is completed for the triggered event. Click **All Assignments** in **CPRS Assignment** to view the list of all computed assignments. For more information, see "CPRS Assignments Table" on page 184.

NOTE: The time taken for computation depends on the number of assignments present in the connected application and Resource Catalog.

---

4 Click to assign or revoke assignments to Resource Catalog.

NOTE: Ensure that the assignment is associated with a resource.

---

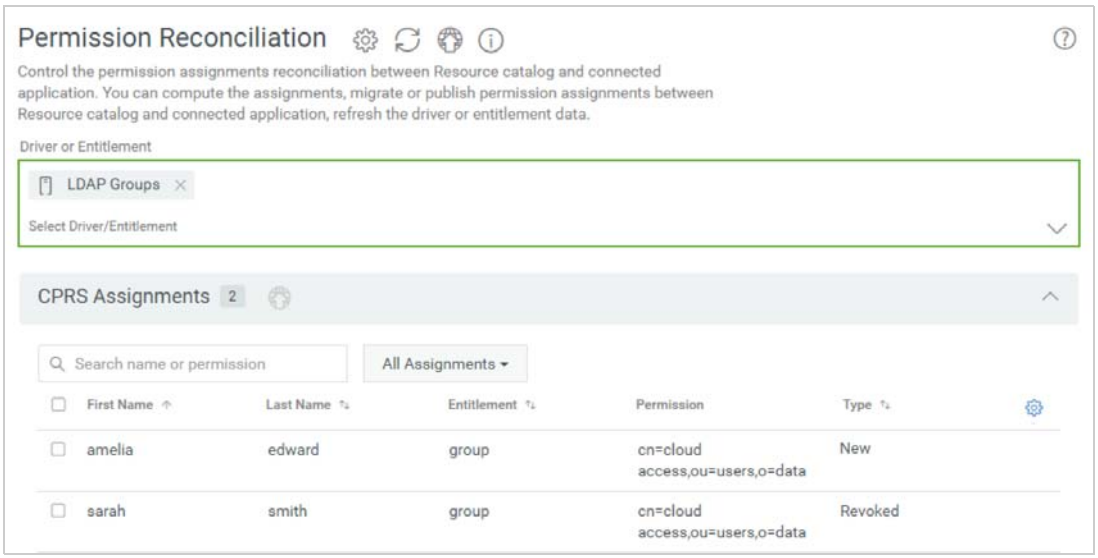
5 (Conditional) Click ⓘ to view the process status of the selected entitlement.

The **PROCESS STATUS** page lists the following columns:

Column Name	Description
Process Type	Specifies the type of processes that are initiated for the entitlement such as Compute or Publish
Start Time	Specifies the start time of the process
Completion Time	Specifies the completion time of the process
Status	Specifies the status of the process. For example, Submitted, In Progress, Completed, or Error
Message	Displays error messages (if any)

CPRS Assignments Table

On selecting an entitlement in the **Permission Reconciliation** page, the assignments appears. If the computation is already performed, all the assignments is displayed.



The following actions can be performed in the **CPRS Assignment** section:

- From the list of displayed assignments, you can filter assignments based on name or permission.
- View the assignments using the following options:
  - All Assignments: This option is selected by default. All the permissions (new and revoked) are displayed.

- New Assignments: This option displays the permissions that are available in the application but not present in the Resource Catalog.
- Revoked Assignments: This option displays the permissions that are present in Identity Manager resource catalog but not in the application.

- To publish one or more assignments to Resource Catalog, select the permission and click  beside **CPRS Assignments**.

NOTE: By default, events generated by CPRS assignments do not flow to the Subscriber channel of the driver. This behavior is controlled by **Allow Entitlement event loopback from cprs to subscriber channel** Engine Control Value. To change the default setting, change the control to **True**. For more information about Engine Control Values, see Engine Control Values in the *NetIQ Identity Manager Driver Administration Guide*.

Migrating to CPRS

Migration to CPRS does not change the resource settings. It only changes the mode of permission reconciliation.This section explains how to migrate resource configurations to CPRS.

- "Prerequisites" on page 185
- "Managing Existing Permissions for AD and LDAP Drivers" on page 186
- "Managing Permissions for a MDAD Driver" on page 186
- "Post Migration Activities" on page 187

Prerequisites

Before migrating the resources, review the following considerations:

- Upgrade from Identity Manager Engine 4.6.x to 4.7

  For more information, see Upgrading Identity Manager Engine (Linux) or Upgrading the Identity Manager Engine (Windows) based on your platform

- Upgrade Identity Applications from 4.6.x to 4.7

  For more information, see Upgrading Identity Applications (Linux) or Upgrading Identity Applications and Identity Reporting (Windows) based on your platform.

- Upgrade the driver packages.

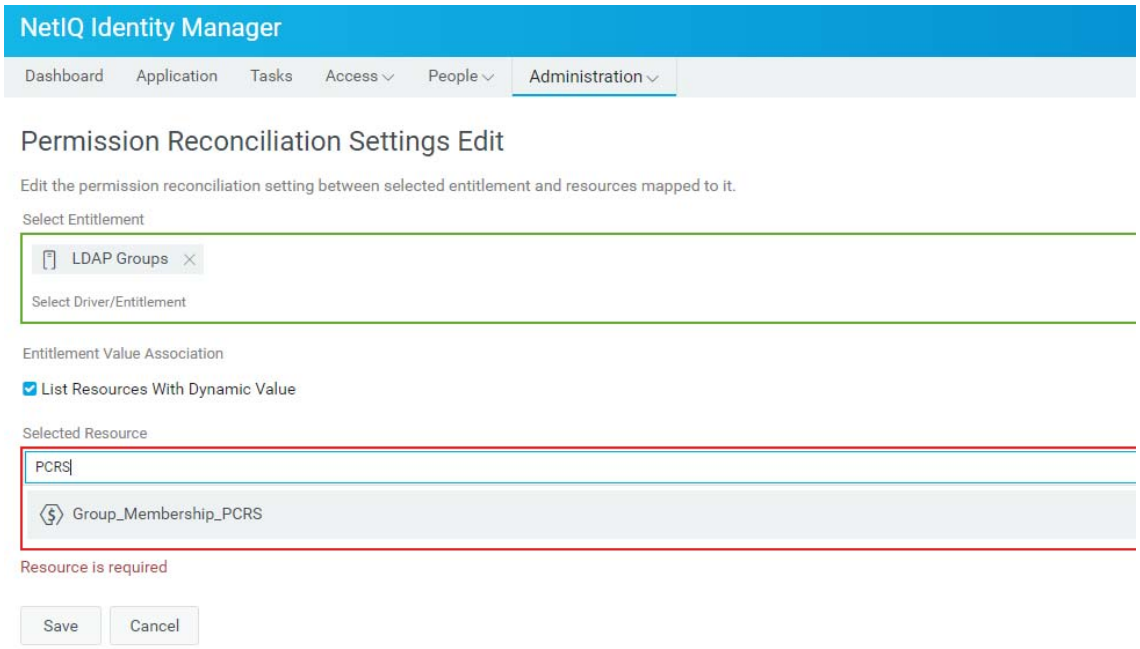
  NOTE: The existing MDAD resources become invalid after the driver is upgraded.

  For more information, see Upgrading the Driver Packages for Identity Applications (Linux) or Upgrading the Driver Packages for Identity Applications (Windows) based on your platform.

Managing Existing Permissions for AD and LDAP Drivers

Managing existing permissions involves migrating the existing resources and creating CPRS settings for those resources in the identity applications. The procedure is similar for AD and LDAP drivers. The following procedure uses LDAP driver as an example.

1. Navigate to **Administration** > **Configuration** > **Permission Reconciliation** and enable **Permission Reconciliation**.
2. In the **Permission Reconciliation Settings Edit** page, select an entitlement. For example: `LDAP_Groups`.
3. Select an existing resource. For example: `Group_Membership_PCRS`.



NOTE: You can select one or more resources for a multivalued entitlement.

4. Click **Save**.
5. Compute and publish permissions for `Group_Membership_PCRS` entitlement.

Managing Permissions for a MDAD Driver

1. Set **Add Logical System information to Entitlement Values** to **Yes** in **Global Configuration Values** using iManager or Designer.

   Enabling this option makes all the existing resources for an entitlement invalid. Therefore, you need to recreate the resources and publish the assignments.
2. Navigate to **Administration** > **Resource** and create new resources that have **Logical System** with entitlements.
3. In the **Permission Reconciliation Settings Edit** page, select an entitlement. For example, `MDAD_Groups`.

4 Select a **Logical System** and map the newly created resource with the new entitlement values.

NOTE: You can select one or more resource for a multivalued entitlement.

5 Click **Save**.

6 Compute and publish permissions for `MDAD_Groups` entitlement.

Post Migration Activities

Few eDirectory objects created during PCRS are not cleaned up during CPRS package upgrade. Manually remove the following eDirectory objects from the driver object path after entitlement package upgrade:

- `PermissionOnboarding`
- `Group_values`
- `PermissionEntMapping`
- `PermissionNameToFile`
- `StaticValueEntitlementMap`
- `EntitlementLLIDMapping` (Only for MDAD)

TIP: Use `idapps.out` and `driverset log` files to trace the CPRS actions and events.

19 Configuring Identity Applications Default Settings

To modify the default settings of identity applications administration settings.

Navigate to **Configuration**.

The settings and configurations made on this page affects while performing any operations on the components that are listed in this page.

- "Configuring Roles and Resources Settings" on page 189
- "Configuring Delegation and Proxy Settings" on page 191
- "Configuring Permission Reconciliation Settings" on page 191
- "Configuring Logging Settings" on page 192
- "Configuring Caching and Cluster Settings" on page 193
- "Assigning Administrators in Identity Applications" on page 199
- "Configuring Workflow Engines and Cluster Settings" on page 204
- "Viewing User Application Driver Status" on page 207
- "Configuring the Default Provisioning Display Settings" on page 207

Configuring Roles and Resources Settings

The **Roles and Resources** page allows you to modify the basic configurations of Roles and Resources Subsytem.

NOTE: You should have both **Role Administrator** and **Resource Administrator** permissions to modify the role and resource configurations on this page.

A **Role Manager** or **Resource Manager** can also access this page. Additionally, they should have **Configuration Role Settings** and **Configuration Resource Settings** permissions to modify the settings. For more information, see "Assigning Permissions to a Delegated Administrator" on page 201.

Configuring Default Roles Settings

The **Role Container**, **Role Request Container**, and **Default Role Approval Definition** show the LDAP settings that are saved in the Identity Vault during installation.

Role Container

The container where all the roles are stored.

Role Request Container

The container where all the role provisioning requests are stored.

Default Role Approval Definition

This determines the default workflow used for role assignment or revocation process.

Role Assignment Grace Period

Set the grace period which determines the time difference between removing the role assignment and dissociating entitlements from the role.

Role Level Display Names

You can change the display names of role levels for all supported languages. To change the language, see "Understanding Roles" on page 24.

NOTE: In identity applications, you can set a role to any of these levels:

- Level 10
- Level 20, or
- Level 30

These levels appear on the roles with their specified display names.

Click **Apply** to save your changes.

Configuring Default Resource Settings

You can view the resource settings that are stored in Identity Vault.

Resource Container

The container where all the resources are stored.

Resource Request Container

The container where all the resource provisioning requests are stored.

Default Resource Approval Definition

The container where all the workflows related to resource approval process is stored. When you select Custom approval process for any resource, it populates the workflow options from this container.

Configuring Entitlement Query Settings

The identity applications periodically make queries to an entitlement from connected systems that are displayed in the **Administration > Resources** list. **Entitlement Query Settings** allow you to specify the interval to refresh the code map tables and also allow you to refresh manually.

Default Query Timeout

Specifies the interval in minutes that system should wait for the query result.

Default Refresh Rate

Specifies the interval in minutes to refresh entitlement queries in the system.

Refresh Status

Indicates whether the entitlement values have been refreshed.

You can refresh **All Drivers** at a time or select specific driver or entitlements that you want to refresh. To refresh the entitlement values manually, click  .

Click **Apply** to save your changes.

Configuring Separation of Duties Settings

You can control the behavior of the separation of duties used in identity applications.

SoD Container

The container where all the SoD constraints are stored.

SoD Approval Definition

To allow permissions for users despite SoD constraints require an approval. This determines the workflow that is used for custom approvals. You can set the approval definition for custom approval process.

This list displays the SoD approval definitions created using Designer. For more information, see *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.

Default Approval Type

This determines the default approval type for SoD constraints when the approval process is enabled for those SoD constraints.

Default SoD Approvers

This determines the default users, groups, roles, or containers who review SoD constraints and approve those requests as required.

Click **Apply** to save your changes.

Configuring Delegation and Proxy Settings

The **Delegation and Proxy** option allows you to configure the retention time for these assignments and the notification template which is used to generate alerts for the respective users. This option also allows you to configure the synchronization and cleanup services for these assignments.

Configuring Permission Reconciliation Settings

The **Permission Reconciliation** page allows you to configure default operations of identity application components. For example, you can set the polling time for status checker and set time for retention of computed permission assignments.

Perform the following actions to configure permission reconciliations:

1 Enable **Permission Reconciliation**, to start using CPRS.

This option helps you to create custom entitlements for connected system roles or resources. You can synchronize the connected application's permission assignments to the Identity Manager resources. To view system resources, go to **Administration > Resources**. For more information about resources, see Chapter 15, "Creating and Managing Resources," on page 163.

2 In **Polling time for status checker (minutes)**, specify the time interval to check the permission reconciliation status.

   By default, the value is set to 10 minutes.

3 In **Retention time for computed permission assignments (days)**, specify the number of days to retain the CPRS process records and computed records.

   By default, the value is set to 7 days.

4 Click **Submit**.

Configuring Logging Settings

Logging allows you to debug the identity applications configuration. The logging service provides facilities for writing, viewing, filtering, and listening for log messages.

By default, Identity Manager saves the logging configuration in `idmuserapp_logging.xml` file that is located in the following location:

`/opt/netiq/idm/apps/tomcat/conf/`

For more information, see Chapter 8, "Setting Up Logging in the Identity Applications," on page 67.

Configuring Auditing Service Settings

Auditing Configuration allows you to enable or disable naudit service and CEF format. To use CEF format, you should specify the following auditing server details after enabling CEF format:

Fields	Description
Destination host	Specifies the destination hostname or IP address of the auditing server.
Destination Port	Specifies the destination port number of the auditing server.
Network Protocol	Specifies the protocol that should be used to establish communication with the auditing server. To establish a secure communication with the auditing server, select **TCP** protocol and enable **Use TLS** option.
Intermediate event store directory	Specifies the temporary directory where the events can be are stored. This directory serves as a backup for an auditing server. **NOTE:** Ensure that the `novlua` permissions are set for the intermediate event store directory. Otherwise, you cannot access the Identity Applications page. To change the permission and ownership of the directory, run the `chown novlua:novlua /<directorypath>` and `chmod 755 /<directorypath>` commands, where `<directorypath>` is the intermediate cache file directory path. Restart Tomcat for the changes to take effect.

Configuring the Identity Manager Packages and their Log Levels

Each feature in identity applications uses one or more packages. Each package handles a specific area of a feature and has its own independent log level that obtains event messages from different parts of the application.

The package names are based on log4j conventions. The event messages include these package names indicating the context of the message output. The logs include tags and values that allow the administrator to identify and correlate which package log entries pertaining to a given transaction and user.

The logs contain information about processing and interactions among identity applications components that occur while fulfilling users and administrative requests and during general system processing. By enabling the correct log levels for various packages, an administrator can monitor how identity applications process users and administrative requests. For more information, see "Configuring Logging Settings in Identity Manager Dashboard" on page 85.

Configuring Caching and Cluster Settings

Caching allows you to manage various caches maintained by Identity Applications. These caches store the reusable data temporarily on the application server to optimize the system performance.

This page displays the cache settings (latest to your application restart). You can manage the cache collection mechanism by changing their configuration settings. You can also flush the cache contents, if necessary.

- "Flushing Caches" on page 193
- "Configuring Cache Settings" on page 194
- "Managing Cluster Cache Settings" on page 199

Flushing Caches

The caches are named according to the subsystems that use them in the Identity Manager User Application. Normally, you don't need to flush them yourself, because the User Application does that automatically based on how frequently their data is used or when the source data changes. However, if you have a specific need, you can manually flush selected caches or all caches.

1. Go to **Configuration > Caching and Cluster**.
2. In **Flush Cache**, select the type of cache from the list that you want to flush.
3. Click **Flush Cache**.

Flushing the Directory Abstraction Layer Cache

The Identity Applications directory abstraction layer also has a cache. The `DirectoryAbstractLayerDefinitions` cache stores abstraction layer definitions on the application server to optimize performance for all data model operations.

In a typical situation, the Identity Applications automatically keeps the `DirectoryAbstractLayerDefinitions` cache synchronized with the abstraction layer definitions stored in the Identity Vault. But, if necessary, you can manually flush the `DirectoryAbstractLayerDefinitions` cache as described in "Flushing Caches" on page 193 to force the latest definitions to be loaded from the Identity Vault.

For more information on the User Application's directory abstraction layer, see NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications.

Flushing Caches in a Cluster

Cache flushing is supported in both clustered and non-clustered application server environments. If your application server is part of a cluster and you manually flush a cache, that cache is automatically flushed on every server in the cluster.

Configuring Cache Settings

You can use the Caching page to display and change cache configuration settings for a clustered or non-clustered application server environment. Your changes are saved immediately, but they don't take effect until the next restart Identity Applications server.

TIP: To restart the Identity Applications, you can reboot the application server; redeploy the application (if the WAR has been changed in some way), or force the application to restart (as described in your application server's documentation).

- "How Caching Is Implemented" on page 194
- "How Cache Settings Are Stored" on page 194
- "How Cache Settings Are Displayed" on page 195
- "Changing Basic Cache Settings" on page 195
- "Changing Non Customizable Cache Settings" on page 196
- "Changing Customizable Cache Settings" on page 196

How Caching Is Implemented

In the Identity Applications, caching is implemented via JBoss Cache. JBoss Cache is an open source caching architecture that's included with the JBoss Application Server but also runs on other application servers.

How Cache Settings Are Stored

Two levels of settings are available for controlling cache configuration: global and local. Use these settings to customize the caching behavior of the Identity Applications.

There are two levels of settings available to control the cache collection on your application server:

- **Global Settings:** Global settings are stored in a central location (the Identity Vault) so that multiple application servers can use the same setting values. For example, If you have a cluster of application servers, the cluster configuration values use the global settings.

To find the global settings in your Identity Vault, look for the following object under your User Application driver:

`configuration.AppDefs.AppConfig`

For example:

`configuration.AppDefs.AppConfig.MyUserApplicationDriver.MyDriverSet.MyOrg`

The XmlData attribute of the configuration object contains the global settings data.

- **Local Settings:** Local settings are stored separately on each application server so that an individual server can override the value of one or more global settings.

  For example, you might want to specify a local setting to remove an application server from the cluster specified in the global settings, or to reassign a server to a different cluster.

  Go to `tomcat/conf/ism-configuration.properties`, the sample local cache settings are:

  ```
  com.sssw.fw.cache.LockAcquisitionTimeout = 15000
  com.sssw.fw.cache.EvictionPolicyClass =
  org.jboss.cache.eviction.LRUPolicy
  com.sssw.fw.cache.eviction.WakeUpIntervalSeconds = 4
  ```

  When you enable the **Local Settings** option for a cache, the modified local settings are stored in the `ism-configuration.properties` file.

The global settings are the default values for every application server that uses a particular instance of User Application driver. Altering the global settings values affects every server unless it specifies local settings to override the global settings.

How Cache Settings Are Displayed

The Caching page displays the current cache settings (from the latest User Application restart). It also displays the corresponding global and local values of those settings and lets you change them (for use at the next User Application restart).

The global settings always have values. The local settings are optional.

Changing Basic Cache Settings

Settings	What to do
Lock Acquisition Timeout	Specify the time interval (in milliseconds) that the cache waits for a lock to be acquired on an object. You might want to increase this setting if the Identity Applications imposes a lot of lock timeout exceptions in the application log. The default value is 15000 ms.

Settings	What to do
Wake Up Interval Seconds	Specify the time interval (in seconds) that the cache eviction policy waits before invoking the following activities: ◆ Processes the evicted node events. ◆ Cleanup the size limit and expired nodes.
Eviction Policy Class	Specify the classname for the cache eviction policy that you want to use. The default is the LRU eviction policy that JBoss Cache provides: `org.jboss.cache.eviction.LRUPolicy` If appropriate, you can change this to another eviction policy that JBoss Cache supports.

TIP: In Local Settings, select Enable Local for the required settings to override the global settings and specify the values.

Changing Non Customizable Cache Settings

Settings	What to do
Max Nodes	Specify the maximum number of nodes allowed in the cache. If you don't want to restrict the number of nodes, specify `0`.
Time To Live Seconds	Specify the time to idle (in seconds) before the node is swept away. If you don't want to restrict the Time To Live Seconds, specify `0`.

TIP: In Local Settings, select Enable Local for the required settings to override the global settings and specify the values.

Click Save to save your configuration values.

Changing Customizable Cache Settings

This allows you to customize certain cache holders in identity applications. To modify the cache holders:

1. Click the Cache Holder ID that you want to modify. For more information about *Cache Holders*, see Table 19-1, "Customizable Cache Holders," on page 197.
2. (Conditional) Change the required values such as Max Nodes, Time To Live Seconds, and Max Age.

   NOTE: The system clears the events in the cache according to the value specified for Max Age.
3. (Conditional) In Local Settings, select Enable Local for the required settings to override the global settings and specify the values.
4. Click Save.

NOTE: You must restart the Tomcat on each node of the cluster for the changes to take effect.

Table 19-1 *Customizable Cache Holders*

Cache Holder Name	Description
Authorization.Admin.Levels CacheHolder	Caches the Administrator type information of the logged in user such as a domain administrator, delegated administrator, team manager, or a business user (self-service). See, "Administrator and Manager Categories" on page 30.
DirectoryAbstractionLayerDefinitions	Caches the Directory Abstraction Layer definitions to optimize performance for all data model operations. See "Flushing the Directory Abstraction Layer Cache" on page 193.
DirectoryService.Container CacheHolder	Caches containers in the directory layer. Containers are shared by many users and groups, and reading them from the directory layer involves both network communication (with the LDAP server) and object creation. By default, the cache is limited to 50 containers, and the LRUs have a default Time To Live (TTL) of 10 minutes. Depending on the directory topography in your enterprise, you might need to adjust the maximum number of nodes or the TTL if you find the performance is suffering because of queries to the LDAP server for container objects. Making settings too high in combination with a large number of usable containers can cause unneeded memory consumption and net lower performance from the server.
DirectoryService.DelProxyRuntimeServiceDelegate	Caches delegate assignments.
DirectoryService.DelProxyRuntimeService.Delegation	Caches user availability settings.
DirectoryService.DelProxyRuntimeService.Delegator	Caches the delegator entities.
DirectoryService.DelProxyRuntimeService.Proxy	Caches proxy assignments.
DirectoryService.GroupCacheHolder	Caches groups in the directory layer. Groups are often shared by many users, and reading them from the directory layer involves both network communication (with LDAP server) and object creation. By default, the cache is limited to 500 groups, and the LRUs have a default TTL of 10 minutes. Depending on the user/group topography in your enterprise, you might need to adjust the maximum number of nodes or the TTL if you find the performance is suffering because of queries to the LDAP server for groups objects. Settings that are too high, in combination with a large number of usable groups, can cause unneeded memory consumption, and net lower performance from the server.

Cache Holder Name	Description
DirectoryService.MembershipCacheHolder	Caches the relationship between a user and a set of groups. Querying the set of groups a user belongs to can be a network and CPU intensive operation on the LDAP server, especially if dynamic groups are enabled. For this reason, relationships are cached with an expiration interval so that changes in the criteria for inclusion/exclusion in a group (such as time-based dynamic groups) are reflected. The default Max Age is five minutes. However, if you use dynamic groups which have a requirement for finer grained time control, then you can adjust the Max Age on this cache holder to be just below the minimum time your finest grained time based dynamic group requires. The lower this value is, the more times the user's groups are queried during a session. Setting a value too high keeps the user/group relationships in memory perhaps longer than the user's session needlessly consuming memory.
DirectoryService.RolesMembershipCacheHolder	Caches the application role membership list by role.
DirectoryService.TeamManagerRuntime.Team	Caches the application team instances and team provisioning requests.
DirectoryService.UserCacheHolder	Caches users in the directory layer. Reading users from the directory layer involves both network communication (with LDAP server) and object creation. By default, the cache is limited to 1000 users, and the LRUs have a default TTL of 10 minutes. Depending on the user topography in your enterprise, you might need to adjust the maximum number of nodes or the TTL if you find the performance is suffering because of queries to the LDAP server for user objects. Making settings too high combined with a large number of different users logging in can cause unneeded memory consumption and net lower performance from the server.
GlobalCacheHolder	The general purpose cache holder. This configuration applies to all caches that are not customizable (that is, all cache holders not listed in this table.)
JUICE	Caches the resource bundles used by the user interface controls and DN display expression lookup results. Changing the setting of the cache holder has a performance impact for the DN display expression lookups because they are frequently used in the User Application. The low value should be at least 300 seconds, but a higher value than 900 seconds is ok. A lower value should be used if the customer is frequently changing the attributes that are used in the DN display expression
RoleManager.RolesCacheHolder	Caches user role memberships listed by the user.
Workflow.Model.Process	Caches the provisioning process XML object structure.
Workflow.Model.Request	Caches the provisioning request XML object structure.
Workflow.Provisioning	Caches provisioning request instances that have not completed. The default maximum capacity for the LRU cache is 500. The capacity can be modified by clicking the Administration/Provisioning and choosing the Engine and Cluster settings. The Process Cache Maximum Capacity appears on this page. This cache reduces the memory footprint for workflow processing without compromising performance.

Managing Cluster Cache Settings

Specify the following settings in Cluster Configuration that helps in caching across the cluster:

Setting	What to do
Permission Index Cluster Enabled	Enable this option if you want to update the permission index changes to the other nodes in the cluster for the specified **Permission Index Group ID**.
Permission Index Group Id	Specify the Permission Index Group ID of the JGroups cluster in which you want to participate. There's no need to change the default Group ID that's provided for the User Application's cluster unless you want to use a different cluster.
Permission Index Cluster Properties	Specify the JGroups protocol stack for the cluster specified by Permission Index Group ID. This setting is to adjust the cluster properties.
Cluster Enabled	Enable this option if you want to overwrite the cache changes to the other nodes in the cluster for the specified **Group ID**.
Group ID	Specify the Group ID of the JGroups cluster in which you want to participate. There's no need to change the default Group ID that's provided for the User Application's cluster unless you want to use a different cluster. The Group ID must be unique and must not match any of the known JBoss cluster names such as `DefaultPartition` and `Tomcat-Cluster`. **TIP:** To see the Group ID in logging messages, make sure that the level of the caching log (`com.sssw.fw.cachemgr`) is set to Info or higher.
Cluster Properties	Specify the JGroups protocol stack for the cluster specified by Group ID. This setting is to adjust the cluster properties.

TIP: In **Local Settings**, select **Enable Local** for the required settings to override the global settings and specify the values.

Assigning Administrators in Identity Applications

An administrator assignment specifies a domain type (Provisioning, Role, Resource, and Security), as well as a set of permissions for the assignment. For more information, see "Administrator and Manager Categories" on page 30.

To assign administrative roles, you must either be a Security Administrator or have a Domain Administrator-type of role, such as Provisioning Administrator.

NOTE: Delegated administrators (Domain Managers) of a domain have no access to **Administrator Assignments** page.

The permissions for an administrator assignment define the actions that administrators can take on a particular scope of object instances within the domain type selected. For example, if you select the Role domain as the domain type for an assignment, the permissions determine what actions the administrators can take on the set of role instances selected as the scope for the assignment. These permissions might specify, for the selected scope of roles, that administrators can perform actions such as assigning roles to users, viewing role assignments, and deleting on role assignments.

IMPORTANT: *Compliance*, *Configuration*, and *Reports* domain types are discontinued from Identity Manager 4.7.1. This change does not remove the existing assignments that have been previously made to these domain types. However, you cannot edit those assignments.

The *Reports* domain type is deprecated with this release. You must use the *Identity Reporting* functionality to manage Identity Manager reports. This requires you to assign **Reporting Administrator** role to any users that you want to access the reporting functionality. You can assign this role to a user in one of the following ways within the identity applications:

- By requesting **Reporting Administrator** role using the **Request** page. See, Requesting Permissions in *NetIQ Identity Manager - User's Guide to the Identity Applications*.
- By selecting **Reporting Administrator** role and assigning to a user in the **Roles** page. See, "Assigning Roles to Users" on page 157.

- "Listing the Administrator Assignments" on page 200
- "Creating a New Administrator Assignment" on page 200
- "Assigning Permissions to a Delegated Administrator" on page 201
- "Deleting an Administrator Assignment" on page 204

Listing the Administrator Assignments

You can search for administrator assignments by specifying the username. You can also filter the assignments by User, Group, Container, or Role categories.

Creating a New Administrator Assignment

You can create an administrator assignment for a user, group, container, or role type. Perform the following steps to create a new administrator assignment:

1. Click +.
2. Specify the **Initial Request Description** that describes the purpose of this assignment.
3. Select the **Domain Type** from the list.

Domain	Description
Provisioning	This domain defines the rights to launch and retract process requests, manage addressee tasks, and configure delegate, proxy, and availability settings.
Role	This domain defines the rights to manage roles and SoDs, assign, revoke, and report on roles, as well as rights to configure role settings.
Resource	This domain defines the rights to manage resources, assign, revoke, and report on resources, as well as rights to configure resource settings and bind entitlements.
Security	This domain defines the rights to manage Identity Applications security, such as assign and revoke domain administrators and managers. This also provides the right to configure teams.

4 Select the **Assignment Type** for which you want to create an assignment.

   This displays the list of users, groups, container, or roles based on the selected assignment type.

5 Select the required user, group, container or a role on from the provided list to create an assignment.

6 (Conditional) Specify the **Effective Date** for this assignment. If you do not specify any date, creates an assignment immediately.

7 (Conditional) Specify the **Expiration Date** for this assignment. If you do not specify any date, the expiration date is set to never.

8 (Conditional) To create a domain administrator assignment for the selected domain, enable **All Permissions**.

   NOTE: This option cannot be edited after creating the assignment. For a delegated administrator, you can assign permissions individually. See, "Assigning Permissions to a Delegated Administrator" on page 201.

   If this option is disabled, a delegated administrator is created for the selected domain.

9 Click **Create**.

Assigning Permissions to a Delegated Administrator

A delegated administrator has the ability to perform selected operations for a subset of authorized objects within the domain for all users. For more information about different types of users, see "Types of User Categories in Identity Applications" on page 29.

The permissions are displayed for an assignment based on the domain type of the assignment. For more information, see Step 3 in "Creating a New Administrator Assignment" on page 200.

To assign permissions for the assignment, you should select the required permissions from the categories. Following sections explain the permissions associated with the Identity Applications domain types:

- "Provisioning" on page 201
- "Role" on page 202
- "Resource" on page 203
- "Security" on page 204

Provisioning

This domain type consists of the permissions that are related to Provisioning Request Definitions (PRD) and User Application Driver.

Category	Permission
Provisioning Request Definition Permissions	This category allows you to assign any of the following permissions for the selected Provisioning Request Definition to a delegated administrator: ◆ **Initiate PRD:** Allows the user to initiate the selected provisioning requests. **NOTE:** The Initiate PRD permission has no effect on the behavior of the installed PRDs for resources and roles within the Identity Applications since these PRDs cannot be initiated directly from the Identity Applications. However, this permission does control whether these PRDs can be initiated from a SOAP call. ◆ **Retract PRD:** Allows the user to retract the selected provisioning requests when they are in progress. ◆ **View Running PRD:** Allows the user to view the selected provisioning requests when they are in progress. ◆ **Configure Delegate:** Allows the user to configure delegate assignments for the selected provisioning requests. ◆ **Manage Addressee Task:** Allows the user to manage tasks associated with the selected provisioning requests that have been addressed to other users. When this permission is enabled, Domain and Delegated Administrators can manage tasks for all users, including addresses and recipients. Managers are able to manage tasks for addressees, but not for recipients. ◆ **Configure Availability:** Allows the user to configure availability for tasks associated with the selected provisioning requests.
User Application Driver Permissions	This category allows you to assign the **Configure Proxy** permission to the delegated administrator. This permission allows the user to configure proxy assignments for the provisioning requests.

Role

This domain type consists the permissions related to roles, Separation of Duties, and configuration of role settings.

Category	Permission
Role Permissions	This category allows you to assign any of the following permissions for the selected Role Level or Roles to a delegated administrator: ◆ **Create Role:** Allows the user to create roles. ◆ **Delete Role** Allows the user to delete the selected roles. This setting applies only at the container level. At installation time, no user has the ability to delete system roles. However, the administrator may grant the user access to the system roles. ◆ **Update Role and Role Relationship:** Allows the user to update the selected roles and modify role relationships. This setting applies only at the container level. ◆ **View Role:** Allows the user to view the selected roles. This setting applies only at the container level. ◆ **Assign Role to a User:** Allows the user to assign users to the selected roles. **IMPORTANT:** Only the Security Administrator can assign system roles to a user. ◆ **Revoke Role from a User:** Allows the user to revoke user assignments for the selected roles. ◆ **Assign Role To Group and Container:** Allows the user to assign groups and containers to the selected roles. ◆ **Revoke Role From Group and Container:** Allows the user to revoke group and container assignments for the selected roles.
Separation of Duties Permissions	This category allows you to assign any of the following permissions for the selected SoDs to the delegated administrator: ◆ **Create SoD:** Allows the user to create the separation of duties constraints. ◆ **Update SoD:** Allows the user to update the selected separation of duties constraints. ◆ **Delete SoD:** Allows the user to delete the selected separation of duties constraints. ◆ **View SoD:** Allows the user to look at the selected separation of duties constraints.
Configuration Permissions	This category allows you to assign the **Configure Role Settings** permission to the delegated administrator. This permission allows the user to configure the settings of the roles subsystem.

Resource

This domain type consists the permissions related to resources, entitlements, and configuration of resource settings

Category	Permission
Resource Permissions	This category allows you to assign any of the following permissions for the delegated administrator: ◆ **Create Resource:** Allows the user to create resources. ◆ **Delete Resource:** Allows the user to delete the selected resources. ◆ **Update Resource:** Allows the user to update the selected resources. ◆ **View Resource:** Allows the user to view the selected resources. ◆ **Assign Resource:** Allows the user to assign users to the selected resources. ◆ **Revoke Resource:** Allows the user to revoke user assignments for the selected resources. If you want to provide access only for the specific container or resources. You can select **Resource Sub Container** or **Select Resources** and assign the required permissions for the administrator.
Entitlements Permissions	This category allows you to assign the **Bind Entitlement** permissions to the delegated administrator. This permission allows the user to bind entitlements with a resource for the selected drivers.
Configuration Permissions	This category allows you to assign the **Configure Resource Settings** permission to the delegated administrator. This permission allows the user to configure the settings of the resource subsystem.

Security

When you select this domain type all permissions are provided. Therefore, the assignments that belong to this domain type will have **All permissions** enabled at the time of assignment creation.

Deleting an Administrator Assignment

You can delete one or more assignments from the **Administrator Assignments** page. To delete multiple assignments, select multiple check boxes against the required assignments.

Configuring Workflow Engines and Cluster Settings

The **Workflow Engine and Cluster Settings** page helps in configuring the Workflow Engine and configuring cluster settings. These settings apply to all engines in the cluster. When any of these settings are changed, other engines in the cluster will detect these changes in the database and use the latest values. The engines check for changes to these settings at the same rate as specified by the **Pending Process Interval**. For more information about configuring workflow engines settings, see "Configure the Workflow Engine Settings" on page 205.

When the workflow engine starts up it checks to see if its engine ID is already in use by another node in the cluster. When this is the case, the workflow engine checks the cluster database to see if the status of the engine is SHUTDOWN or TIMEDOUT. If it is, the workflow engine starts. If the status is STARTING or RUNNING, the workflow engine logs a warning, then waits for a heartbeat timeout to

occur. If the heartbeat timeout occurs, that means that the other workflow engine with the same ID was not shut down properly, so it's safe to start. If the heartbeat timer is updated, that means another workflow engine with the same ID is running in the cluster, so the workflow engine cannot start. You can specify the heartbeat timeout (the maximum elapsed time between heartbeats before a workflow engine is considered timed out) by setting **Heartbeat Interval** and **Heartbeat Factor**. For more information about configuring the cluster settings, see "Configure Workflow Cluster Settings" on page 206.

The process cache settings and heartbeat settings require a server restart to take effect.

Configure the Workflow Engine Settings

Following are the engine settings that you might require to configure for your workflow engine settings:

Engine Setting	Description
Enable Email Notification	Enables or disables email notifications for the entire workflow engine. Defaults to enabled.
Web Service Activity Timeout (minute)	Specifies the default Web Service activity timeout in minutes. The default is 50 minutes.
User Activity Timeout (hour, 0 for no timeout)	Specifies the default user activity timeout. The default is 0 days, which indicates no timeout.
Completed Process Timeout (day)	Specifies the number of days that a completed process state is kept in the workflow database system. The default is 120 days.
Completed Process Cleanup Interval (hour)	Specifies how often the engine checks for and removes completed processes that have been in the workflow database system for longer than the completed process timeout. The default is 12 hours.
Pending Process Interval (second)	User activities that are executed on an engine which the process is not bound to are put into a pending state. This interval specifies how often to check for pending activities in order to continue their execution. The default is 30 seconds.
Retry Queue Interval (minute)	Activities that fail because of suspected database connectivity issues are put on a retry queue. This interval specifies how often the engine attempts to retry these activities. The default is 15 minutes.
Thread Keep Alive Time (second)	If the pool is larger than the minimum size, excess threads that have been idle for more than the keep-alive time will be destroyed. The default is 5 minutes.
Maximum Engine Shutdown Timeout (minute)	The engine attempts to shutdown gracefully. When shutting down it stops queuing new activities for execution and attempts to complete any activities already queued. This timeout specifies the maximum time that the engine waits for all queued activities and threads executing activities to complete. If this time is exceeded, the engine halts processing of queued activities and attempts to stop all threads executing activities. The default is 1 minute.
Maximum Thread Pool Size	The maximum number of threads that the engine uses to execute activities. The default is 20.

Engine Setting	Description
Minimum Thread Pool Size	The minimum number of threads that the engine uses to execute activities. When a thread is requested and fewer than the minimum are in the pool, a new thread will be created even if there are idle threads in the pool. The default is `10`.
Initial Thread Pool Size	Number of pre-started threads in the pool when it is created. The default is `5`.
Process Cache Load Factor	The load factor specifies how full the cache is allowed to get before increasing its capacity. If the number of entries in the cache exceeds the product of the load factor multiplied by the current capacity, then the capacity is increased. The default is `0.75`.
Process Cache Initial Capacity	The process cache is backed by a hash map. The capacity is the number of buckets in the hash map. The initial capacity is the number of buckets at the time the cache is created. The default is `700`.
Process Cache Maximum Capacity	Before adding a process to the cache, if the number of processes in the cache equals or exceeds the Process Cache Maximum Capacity, the cache attempts to remove the oldest inactive process from the cache. The maximum capacity is a soft limit, so the number of processes in the cache might exceed the Process Cache Maximum Capacity if there are no inactive processes (only active processes) in the cache. The default is `500`.

Configure Workflow Cluster Settings

Following are the settings that you might require to configure for your workflow cluster settings:

Cluster Setting	Description
Heartbeat Interval	Specifies the interval at which the workflow engine's heartbeat is updated. When the workflow engine starts up, it detects if its engine ID is already being used by another node in the cluster and refuses to start if the ID is in use. The User Application database maintains a list of engine IDs and engine states. If an engine crashes and is restarted, its last state in the database indicates that it is still running. Therefore, the workflow engine uses a heartbeat timer, which writes heartbeats at the specified interval, to determine if an engine with its ID is still running in the cluster. If it's already running, it refuses to start. The minimum value for the heartbeat interval is `60` seconds.
Heartbeat Factor	Specifies the factor that is multiplied with the heartbeat interval to arrive at the heartbeat timeout. The timeout is the maximum elapsed time permitted between heartbeats before an engine will be considered timed out. The minimum value for the heartbeat factor is `2`.

Viewing User Application Driver Status

The **Driver Status** page displays the User Application Driver details such as:

Driver Name

Displays the name of the driver in LDAP format. For example:

```
cn=User Application Driver,cn=driverset1,o=system
```

Driver Version

Displays the driver version used in Identity Manager.

Application Revision

Displays the revised version of Identity Applications.

Patch Level

Displays the patch applied for the driver.

Build Revision

Displays the updated build version.

Status

Displays the driver state.

Configuring the Default Provisioning Display Settings

Provisioning Display Settings page controls the behavior of general search results in Identity Manager Dashboard. You can also modify the appearance of **Tasks** and **Request History** page.

- “Managing General Display Settings” on page 207
- “Managing the Appearance of Tasks Page” on page 208
- “Managing the Appearance of Request History Page” on page 209

Managing General Display Settings

Following settings apply for the search results showing on the accessed Identity Applications pages:

Setting	Description
Default number of results displayed per page	Specifies the default number of rows to display on the Identity Manager Dashboard pages. Identity Applications fetch the results for the specified number and stores in the cache and displays to a user on accessing the Identity Applications pages. Each time the user requests to see the next page, another set of rows is returned from the cache. The default value for this setting is 25.
Options for number of results displayed per page	Allows you to specify additional values that the user can select to override the default number of rows displayed on the Identity Applications pages. The list of values you type must be separated by commas and ranging from 1-10000. **NOTE:** The number specified in Default number of results displayed per page is always included in the list of values for the user to select. The default value for this setting is `5,10,25,50,100,500`.

Managing the Appearance of Tasks Page

Field	Description
Select Column to set default sort	By default, the task results in the Tasks page are sorted by Assigned To. You can select a different column from the list to sort the task results. Also, you can sort the results by ascending or descending order. Use Sort by Descending Order to sort the results in descending order. Disabling this option displays the results in ascending order.
Allow user to customize columns	By default, this option is enabled. Disabling this option restricts the user from customizing columns in the Tasks page. ◆ **Available columns:** Displays the columns which are disabled for user customization. ◆ **User default columns:** Displays the columns that are already showing on the Tasks page. ◆ **Available columns for User customization:** Displays the columns that can be customized by users.
Allow user to customize task detail open	By default, this option is enabled. This option allows you to change the preferences of opening the approval form in the Tasks page. Go to Tasks page and click ⚙ to change the preferences. Disabling this option will restrict the system users from changing the preferences of opening the approval form in the Tasks page. However, you can change this preferences in the Settings > Customization page.

Click Save to apply your changes.

Managing the Appearance of Request History Page

Field	Description
Select Column to set default sort	By default, the request statuses in the **Request History** page are sorted by **Request Date**. You can select a different column from the list to sort the results. Also, you can sort the results by ascending or descending order. Use **Sort by Descending Order** to sort the results in descending order. Disabling this option displays the results in ascending order.
Allow user to customize columns	By default, this option is enabled. Disabling this option restricts the user from customizing columns in the **Request History** page. ◆ **Available columns:** Displays the columns which are disabled for user customization. ◆ **User default columns:** Displays the columns that are already showing on the **Request History** page. ◆ **Available columns for User customization:** Displays the columns that can be customized by users.

Click **Save** to apply your changes.

20 Configuring Email-Based Approval

As an administrator, you can configure the identity applications to send an email that notifies users that they have a pending task to approve or reject a permission request.

Administration > Email-based approval

NOTE: Before enabling email-based approvals, ensure that you have configured the provisioning request definitions (PRDs) to support notifications . Also, configure the outgoing mail server. For more information, see the Email Based Approval in *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.

You can configure the following settings to allow users to approve requests on their emails:

Incoming Email Settings

Specifies the server type, SSL settings, port type, and notification templates.

Outgoing Email Settings

Specifies the notification template host, protocol, and SMTP server. This tab allows specifies digital signature settings.

For more information, click ⓘ on the Dashboard.

21 Configuring and Managing Objects for Entities

Any Identity Vault object that you want to search, display, or edit in the Identity Manager User Applications must be defined in the directory abstraction layer. For more information, see About Entities and Attributes in the *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.

Identity Applications allow you to create and modify objects for a specific entity. The configuration of the entities displayed in the **Entities** tab is specific to the **Client** selected.The **Edit** page displays the list of objects for the selected entity.

After an entity is created, it is listed under **Access Settings** in the **Settings** page. It provides control on which users, roles, groups, and containers can view and list the entities. You must have the appropriate permission to create, view, or manage objects. By default, the trustee is the Provisioning Administrator. Ensure that the users specified in **Trustees** are having sufficient Identity Vault rights to perform tasks within the Identity Applications. However, trustees can access the page but operations on the page will fail if they do not have the proper Identity Vault rights.

You can perform the following operations on the Entities page:

- Listing the objects
- Creating an object
- Editing the object
- Deleting an object
- Exporting to CSV

Listing the Objects

The **Entities** page displays all the objects associated with the selected entity. You can search for attributes based on the criteria specified in the **Settings** page. For more information, see **View Attributes** in "Entity Settings" on page 133. If you wish to view columns other than the default columns, you can customize the them by clicking the ⚙ icon.

IMPORTANT: Do not use these[< > , ; \ " + # = / | & ' ! @ $ %] special characters or a whitespace in the search bar to find an object.

Creating an Object

Perform the following actions to create an object:

1 Click the + icon and specify the fields marked with an asterisk (*).

The fields that are displayed are based on the criteria specified on the **Settings** page. For more information, see **View Attributes** in "Entity Settings" on page 133.

2 Enter the details and click **Create**.

IMPORTANT: Do not use these [< > , ; \ " + # = / | & * ' ! @ $ %] special characters or a whitespace in the **ID** field.

Editing an Object

Perform the following actions to edit an object:

1 Select the object from the list and click **Edit**.

The fields that can be edited are based on the criteria specified on the **Settings** page. For more information, see **Editable Attributes** in "Entity Settings" on page 133.

2 Edit the required fields.

3 Click **Save**.

Deleting an Object

To delete any object from the list, select the entity and click the icon. You can delete one or more objects at a time.

Exporting to CSV

Perform the following actions to save one or more objects in a csv file:

1 Select the required object.

2 Click the icon.

3 Click **Save**.

IV Configuring and Managing Provisioning Workflows

These sections describe how to configure and manage provisioning requests and workflows:

- Chapter 22, “Configuring the User Application Driver to Start Workflows,” on page 217
- Chapter 23, “Managing Provisioning Request Definitions,” on page 223
- Chapter 24, “Managing Provisioning Workflows,” on page 231

22 Configuring the User Application Driver to Start Workflows

This section describes the User Application driver and how to configure it to automatically trigger a workflow based on an event in the Identity Vault.

About the User Application Driver

The User Application driver is responsible for starting provisioning workflows and for notifying the User Application of changes in the Identity Vault. For example, when you make changes to the directory abstraction layer using the Designer for Identity Manager. Only the Subscriber channel is used in this driver. The driver processes messages from the Identity Vault to the User Application running on an application server. Although there are events that occur in the User Application that are reported back to the Identity Vault, these events do not flow through the Publisher channel of the User Application driver.

When the application server is started, the driver establishes a session with the application server. The driver sends messages to the User Application running on the application server (for example, “retrieve a new set of virtual directory definitions”).

The source components of the driver include:

- `ComposerDriverShim.jar` – The Composer Driver Shim. It is installed in the `lib` directory `\Netiq\NDS\lib` in Windows or the `classes` directory `/opt/novell/eDirectory/lib/dirxml/classes` in Linux.
- `srvprvUAD.jar` – The Application Driver Shim. It is installed in the `lib` directory `\Netiq\NDS\lib` in Windows or the `classes` directory `/opt/novell/eDirectory/lib/dirxml/classes` in Linux.
- `UserApplicationDriver.xml` – A file that contains configuration data for setting up the new driver. It is installed in the `DirXML.Drivers` directory, which is `\Tomcat\webapps\nps\DirXML.Drivers` in Windows and either `/opt/netiq/eDirectory/lib/dirxml/rules/` or `/var/opt/novell/iManager/nps/DirXML.Drivers` in Linux.

The User Application driver components are installed when you install Identity Manager. Before you can run the Identity Manager User Application, you must add the User Application driver to a new or existing driver set, and activate the driver.

Depending on your work environment, very little configuration of the User Application driver might be required, or you might want to implement a complex set of business rules in the driver policies. The User Application driver provides the same flexible mechanisms for data synchronization as other Identity Manager drivers.

Setting Up Workflows to Start Automatically

Workflows are automatically started when a user starts a provisioning request by requesting a resource. In addition, the User Application driver listens for events in the Identity Vault and, when configured to do so, responds to events by starting the appropriate provisioning workflows. For example, you can configure the User Application driver to automatically start a provisioning workflow if a new user is added to the Identity Vault. You configure the User Application driver to automatically start workflows using Identity Manager policies and rules.

About Policies

You can use filters and policies with the User Application driver in the same way that you can with other Identity Manager drivers. When an event occurs in the Identity Vault, Identity Manager creates an XML document that describes the event. The XML document is passed along the channel to the connected system (in this case, the connected system is the User Application). Filters and policies associated with a driver allow you to define how to respond to the event, and in the process transform that XML document to the format that is expected by the connected system. Identity Manager provides several categories of policies (for example, Event Transformation, Command Transformation, Schema Mapping, Output Transformation) that you can apply, in a prescribed order, to transform the XML document.

This section provides an example of starting a workflow based on events in the Identity Vault. Although any of the policies can be used to trigger a workflow, the example presented in this section demonstrates the easiest and most useful method.

When you create a User Application driver, an Event Transformation Policy is created for use by the driver. The Event Transformation Policy is responsible for creating the XML document that is processed by the remaining Subscriber channel policies.

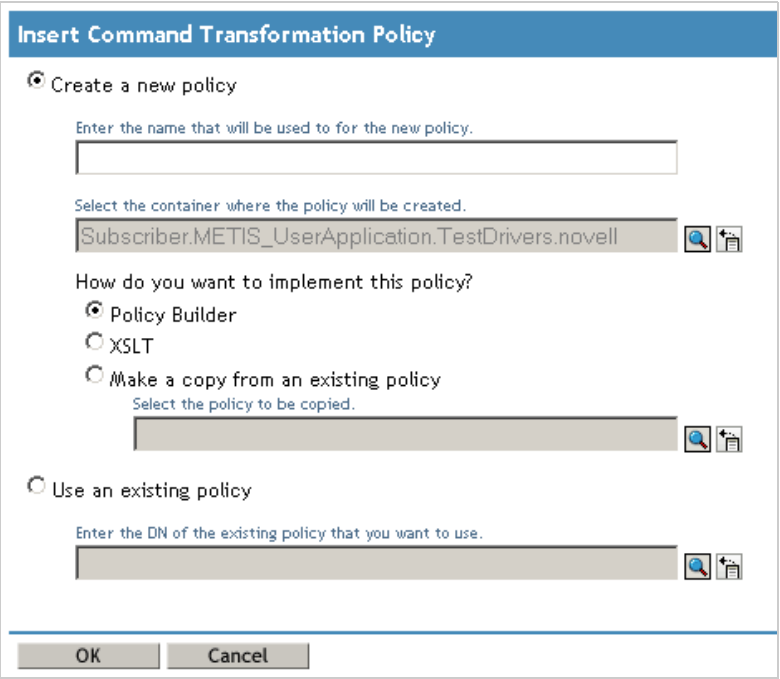
NOTE: Do not change the Event Transformation policy that was created when the User Application driver was created. The DN of this policy begins with `Manage.Modify.Subscriber`. Changing this policy might cause the workflow process to fail.

An empty Schema Mapping Policy is also created. You can use this policy as a starting point for triggering a workflow, based on events in the Identity Vault.

Using the Policy Builder

The Policy Builder provides a Start Workflow action that simplifies the process of setting up a workflow to start automatically.

1. In iManager, expand the **Identity Manager** Role, then click **Identity Manager Overview**.
2. Specify a driver set.
3. Click the driver for which you want to manage policies. The **Identity Manager Driver Overview** opens.
4. Click the policy that you want to edit.
5. Click **Insert** to open the Policy Builder.



6 Click **Create a new policy**.

7 Type a name for the policy.

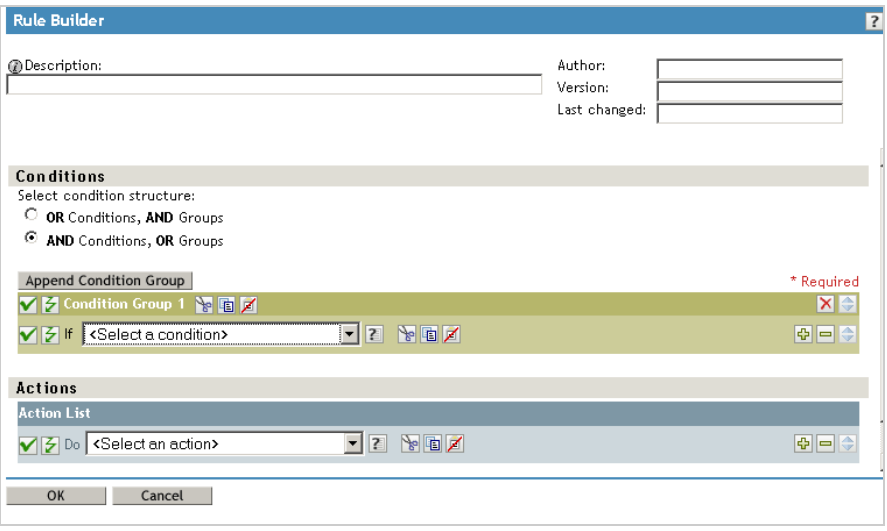
8 Click **Policy Builder**.

9 Click **OK**.

iManager displays a screen that lists defined policy rules.

10 Click **Append New Rule**.

iManager displays the **Rule Builder**.

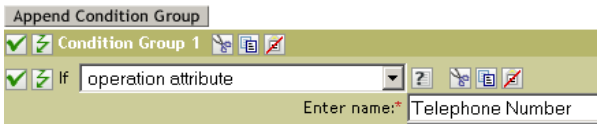


11 Type a **Description** for the rule.

12 Select **operation attribute** for the **If** condition in **Condition Group 1.**

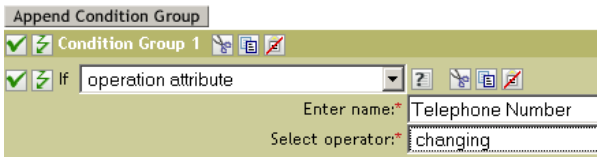
13 Use the **Browse attributes** button for the **Enter name** field to specify the Identity Vault attribute that you want to use to start the workflow.

For example, to start a workflow when a telephone number changes, select the **Telephone Number** attribute.



14 Use the **Select Operator** list to select the operator to use to test the specified attribute.

For example, to start a workflow when a telephone number changes, select **changing**.

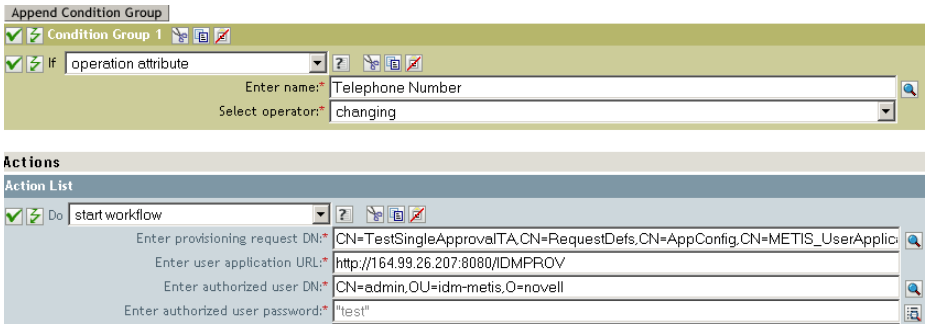


15 Select **start workflow** from the **Action** list.

16 Use the Object Selector in the **Enter provisioning request DN** field to select the provisioning request definition that you want to be executed when the **if** condition is true.

The **Enter user application URL** and **Enter authorized user DN fields** are filled in automatically.

17 Type the password for the User Application administrator in the **Enter authorized user password** field.

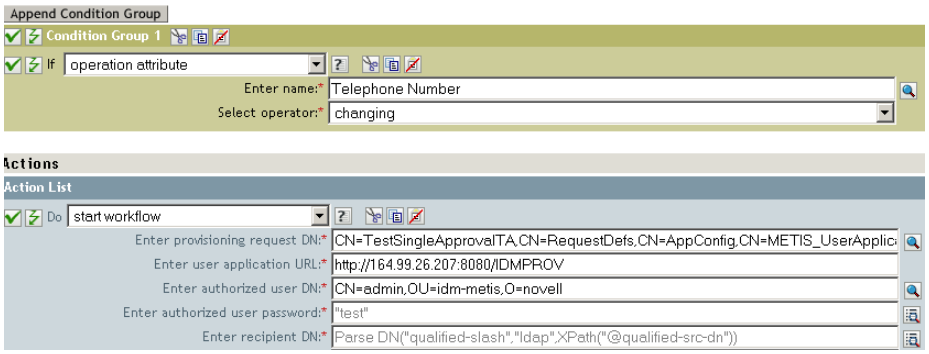


We recommend using a named password, because typing a password in clear text is a security risk.

18 In the **Enter recipient DN** field, specify the DN of the recipient of the workflow in LDAP format.

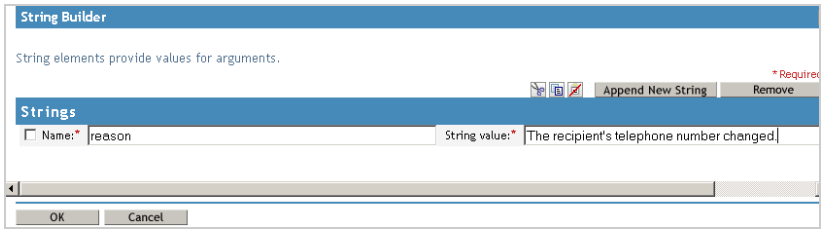
The expression for the recipient DN must evaluate to a DN that conforms to RFC 2253 format (in other words, cn=user,ou=organizational unit,o=organization). For example, you can click the **Argument Builder** button in the **Enter recipient DN** field to create the following expression to pass the recipient's DN to the workflow:

```
Parse DN("qualified-slash","ldap",XPath("@qualified-src-dn"))
```



19 Specify the arguments for the workflow in the **Enter additional arguments** field.

You must use this field to specify the **reason** attribute, which is required by the workflow. You can click the **String Builder** button in the **Enter additional arguments** field to specify the **reason** attribute and create a value for the attribute (for example, “the recipient’s telephone number has changed”).



20 Click **OK** to close the Rule Builder.

21 Click **OK** to close the Policy Builder.

22 Click **OK** to close the Policies screen.

23 Make sure that you add any attributes needed by the workflow to the filter.

In the example described in this procedure, you would need to add **Telephone Number** and **CN** to the filter.

23 Managing Provisioning Request Definitions

This section provides instructions for managing provisioning request definitions.

About the Provisioning Request Configuration Plug-in

You can use the Provisioning Request Configuration plug-in to iManager to view a read-only display of a provisioning request definition that was created in the Designer for Identity Manager. This plug-in allows you to delete, activate, inactivate and retire existing provisioning request definitions.

NOTE: The Provisioning Request Configuration plug-in to iManager does not allow you to create or edit provisioning request definitions. To create or edit a provisioning request definition, you need to use the Designer for Identity Manager.

You can find the Provisioning Request Configuration plug-in in the Identity Manager category in iManager. The plug-in includes the Provisioning Requests task in the Provisioning Configuration role. The Provisioning Requests task consists of the panels described in Table 23-1.

Table 23-1 *Provisioning Requests Task: Panels*

Panel	Description
Provisioning Driver Selection	Gives you the opportunity to select an Identity Manager User Application driver. The driver contains a set of predeployed provisioning request definitions, so you need to pick a driver before you can begin configuring your provisioning requests.
Provisioning Request Configuration	Provides tools that let you: ◆ Browse the available provisioning request definitions and select one to configure ◆ Create a new provisioning request definition based on an existing definition ◆ Set the properties of a provisioning request definition ◆ Assign the provisioning request definition to a provisioned resource ◆ Edit the addressee and timeout settings for each activity in the associated workflow When you choose to create a new provisioning request or edit an existing one, the plug-in runs the Provisioning Request Configuration Wizard.

Working with the Installed Templates

You can define provisioning request definitions from scratch in the Designer for Identity Manager. Alternatively, you can define provisioning requests by modeling them after the provisioning request templates that ship with the product. To use the templates, 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:

- No approvals
- One approval step
- Two approval steps
- Three approval steps
- Four approval steps
- 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.

Identity Manager ships with the templates listed in Table 23-2.

Table 23-2 *Templates for Provisioning Requests*

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

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

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

Workflows and provisioned resources. When you create a new provisioning request definition, you bind it to a provisioned resource. You can change the provisioned resource associated with the request definition, but not the workflow or its topology.

Categories for provisioning requests. Each provisioning request template is also bound to a category. Categories provide a convenient way to organize provisioning requests for the end user. The default category for all provisioning request templates is **Entitlements**. The category key, which is the value of the srvprvCategoryKey attribute, is **entitlements** (lowercase).

You can create your own categories by using the directory abstraction layer editor. When you create a new category, make sure the category key (the value of srvprvCategoryKey) is lowercase. This is necessary to ensure that categories work properly in the Identity Manager User Application.

For details on creating provisioning categories, see the *Identity Manager User Application: Design Guide*.

Configuring a Provisioning Request Definition

Before configuring a provisioning request definition, you need to select the User Application driver that contains the definition. Having selected the driver, you can create a new provisioning request definition or edit an existing definition. You can also delete provisioning request definitions, change the status of a request definition, or define rights for a request definition.

Selecting the Driver

To select a User Application driver:

1. Select the **Identity Manager** category in iManager.
2. Open the **Provisioning Request Configuration** role.
3. Click the **Provisioning Requests** task.

   iManager displays the User Application Driver panel.
4. Specify the driver name in the **User Application Driver** field, then click **OK**.

   iManager displays the Provisioning Request Configuration panel. The Provisioning Request Configuration panel displays a list of available provisioning request definitions.

   The installed templates appear in dark text with a status of **Template**. Request definitions that are templates do not display hypertext links because they are read only.

   NOTE: If the request definitions were configured to use localized text, the names and descriptions for these definitions show text that is suitable for the current locale.

Changing the driver. When you have selected a driver, the driver selection remains in effect for the duration of your iManager session, unless you select a new driver. To select a new driver, click the **Actions** command, then choose **Select User Application Driver** from the **Actions** menu.

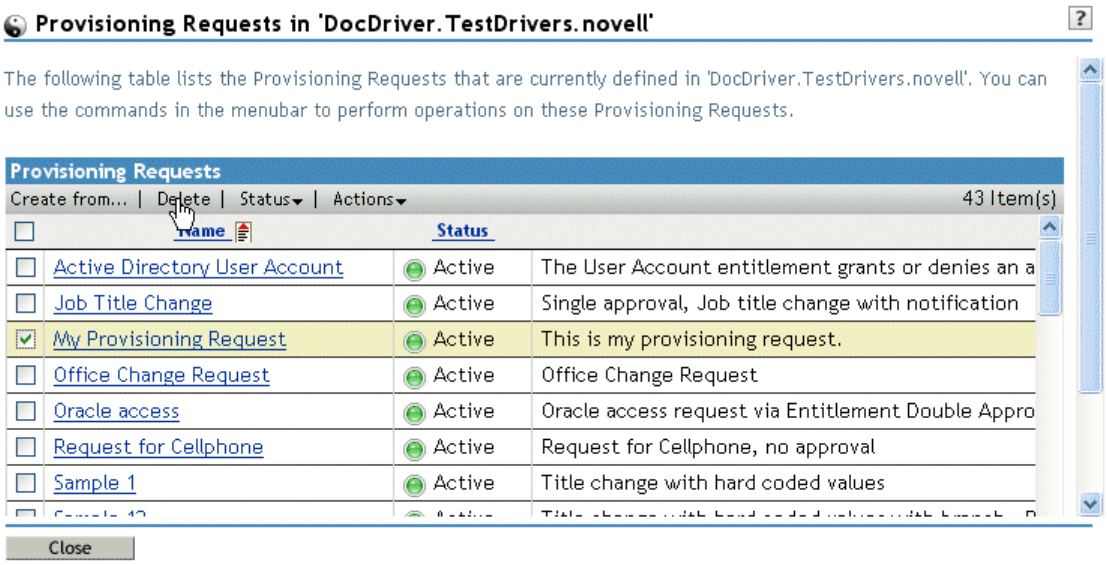
Deleting a Provisioning Request

To delete a provisioning request:

1. Select the provisioning request you want to delete by clicking the check box next to the name.

   You are not permitted to delete a provisioning request that is a template.

2 Click the **Delete** command in the Provisioning Request Configuration panel.

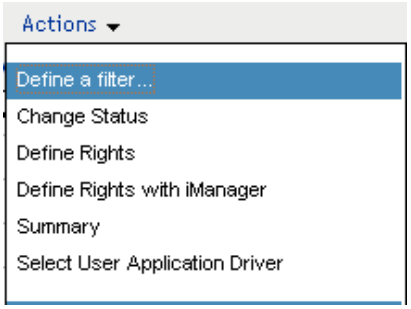


Filtering the List of Requests

To filter the list of requests:

1 Click the **Actions** command in the Provisioning Request Configuration panel.

2 Click the **Define a Filter** command on the **Actions** menu.



Specify the filter characteristics:

Choice	Description
Turn off filtering	Disables any existing filtering for the list.
Filter for status equals	Filters based on the status. You can filter the list based on any of the following status codes: **Active** **Inactive** **Template** **Retired**
Filter for category equals	Filters based on category. Select any of the defined categories.
Filter for description contains	Allows you to search for text in the request description. Type the string you want to search for.

Changing the Status of an Existing Provisioning Request

To change the status of an existing provisioning request:

1. Select the provisioning request for which you want to change status by clicking the check box beside the name.
2. Click the **Actions** command in the Provisioning Request Configuration panel.
3. Click the **Change Status** command on the **Actions** menu.
4. Click the status in the **Status** menu:

Status	Description
Active	Available for use.
Inactive	Temporarily unavailable for use.
Retired	Permanently disabled.

5. Click the button for the correct action (**Grant** or **Revoke**).
6. Click **Finish**.

Defining Rights on an Existing Provisioning Request

To define rights on an existing provisioning request:

1. Select the provisioning request for which you want to define rights by clicking the check box beside the name.
2. Click the **Actions** command in the Provisioning Request Configuration panel.
3. Click the **Define Rights** command on the **Actions** menu.
4. Specify the rights for the request.

To define rights on a provisioning request with iManager:

1. Select the provisioning request for which you want to define rights by clicking the check box beside the name.
2. Click the **Actions** command in the Provisioning Request Configuration panel.
3. Click the **Define Rights with iManager** command on the **Actions** menu.

24 Managing Provisioning Workflows

This section provides instructions for managing provisioning workflows at runtime. It also provides instructions for configuring email notification for provisioning workflows.

About the Workflow Administration Plug-in

The Workflow Administration plug-in to iManager provides a browser-based interface that lets you view the status of workflow processes, reassign activities within a workflow, or terminate a workflow in the event that it is stopped and cannot be restarted.

You can find the Workflow Administration plug-in in the Identity Manager category in iManager. The plug-in includes the **Workflows** task in the **Workflow Administration** role.

The Workflow Administration role also includes the **Email Templates** and **Email Server Options** tasks. These tasks are shortcuts to other tasks listed under the Passwords role.

The Workflows task comprises the panels listed in Table 24-1.

Table 24-1 *Workflows Task: Panels*

Panel	Description
Workflows	Provides the primary user interface for administering provisioning workflows. The interface lists workflows currently being processed and lets you perform various actions on these workflows. When you first start the Workflows task, the Workflows panel requires that you select a User Application driver. The driver points to a workflow server. You need to select a driver before you can log in to the server and begin workflow administration. When you have selected a driver, you can specify search criteria for selecting the workflows to manage.
Workflow Detail	Provides a read-only user interface for viewing the details about a specific workflow.

Managing Workflows

This section includes procedures for managing provisioning workflows using the Workflow Administration plug-in.

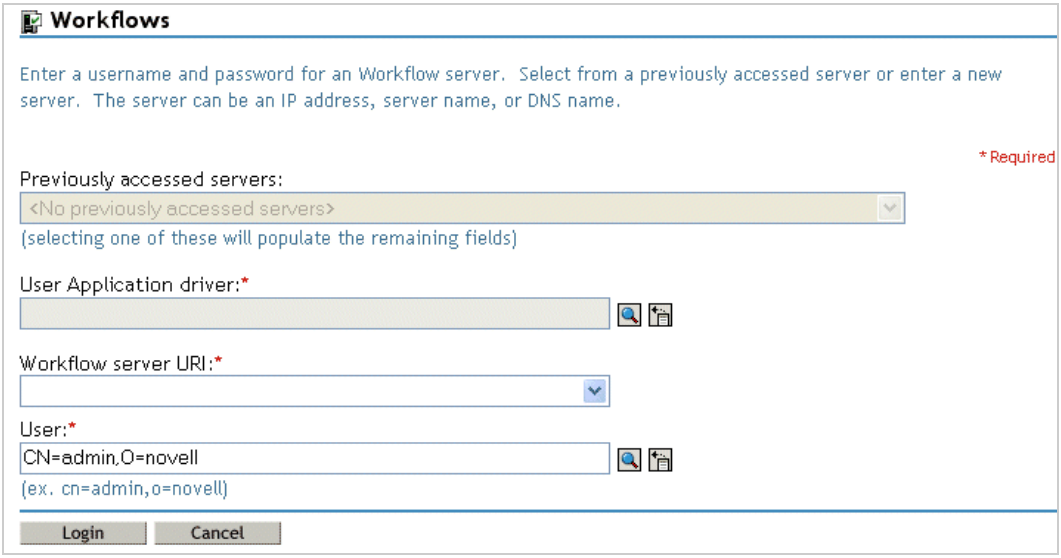
Connecting to a Workflow Server

Before you can begin managing workflows, you need to connect to a workflow server. If the User Application driver is bound to a single workflow server, you can simply specify the name of the driver to use. If the driver is associated with multiple workflow servers, you need to select the target workflow server.

To connect to a workflow server:

1. Select the Identity Managercategory in iManager.
2. Open the **Workflow Administration** role.
3. Click the **Workflows** task.

   iManager displays the Workflows panel.

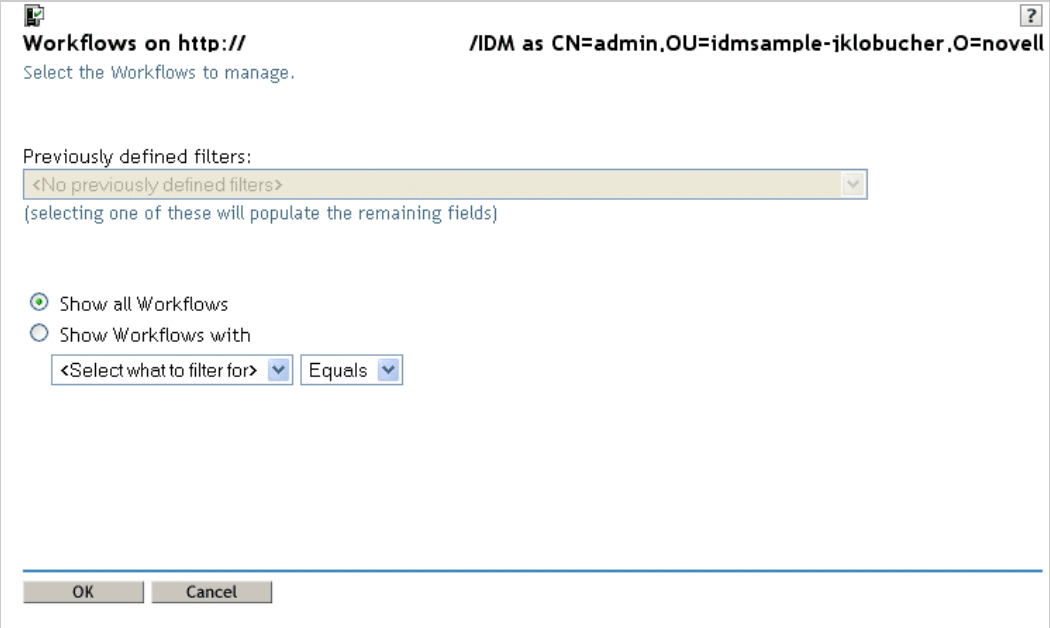


4. If you accessed the target workflow server previously, you can select the server from the **Previously accessed servers** drop-down list.

   iManager fills in the remaining fields on the panel.
5. If you have not yet accessed a workflow server, specify the driver name in the **User Application Driver** field, then click **OK**.

   iManager fills in the **Workflow server URI** and **User** fields.
6. Type the password for the user in the **Password** field.
7. Click *Login*.

The Workflow Administration plug-in displays a page that allows you to specify a filter for finding workflows:



Restricting Access to Workflows

When you install the Roles Based Provisioning Module with the default settings, all users can view all workflows when searching for them. To restrict access to workflows to only the workflow trustees, use NetIQ eDirectory to add an Inheritance Rights Filter (IRF) on the `AppConfig` container, which is located under the User Application driver. For information about adding an IRF, see the eDirectory Administration Guide.

Finding Workflows that Match Search Criteria

If the target workflow server is running a large number of workflow processes, you might want to filter the list of workflows you see in iManager. To do this, you can specify search criteria.

1. Select **Show Workflows with**.

   By default, **Show all Workflows** is selected. Do not change the default if you want to see the complete list of workflows on the server.
2. Select the attribute for which you want to specify criteria.

Attribute	Description
Creation time	Time that the workflow was initiated.
Initiator	Username of the requestor.
Recipient	Username of the recipient.
Process Status	Status of the workflow process as a whole (Completed, Running, or Terminated).
Approval status	Status of the approval process (Approved, Denied, or Retracted).
Entitlement status	Status of the entitlement initiated by the provisioning request (Error, Fatal, Success, Unknown, or Warning).

3 Select an operator:

Operator	Comment
Equals	Supported for all attributes.
Before	Only supported for the Creation time attribute.
After	Only supported for the Creation time attribute.
Between	Only supported for the Creation time attribute.

4 Specify a value in the field below the attribute and operator.

For **Creation time**, you can use the **Date and time** control to select the value. For **Initiator** and **Recipient**, you can use **Object History** or **Object Selector** to specify a value. For all other attributes, select the value from the drop-down list.

5 Click **OK**.

iManager displays the workflows you have selected on the Workflows panel.

Changing the target server and filter. When you have selected a workflow server, this selection remains in effect for the duration of your iManager session, unless you select a new server. To select a new server, click the **Actions** command, then choose **Select Server** from the **Actions** menu.

To specify different search criteria, choose *Define Filter* on the **Actions** menu.

Controlling the Active Workflows Display

The Workflows panel lists the workflows that match the search criteria you specified. In addition to filtering the list, you can control the display. For example, you can specify how often to refresh the list and sort the list on a particular column.

Refreshing the List of Workflows

When the workflow server is very busy, the list of active workflows can change very frequently. In this case, you should refresh the list of active workflows running on the server.

1. Click the **Refresh** command in the Workflows panel.
2. Specify the refresh interval you want to use by selecting one of these options from the **Refresh** menu:
   - Refresh Off
   - Refresh Now
   - 10 seconds
   - 30 seconds
   - 60 seconds
   - 5 minutes
3. Click **OK**.

Using Quick Filters to Control the Display

Sometimes you might want to show or hide workflows that have a particular status.

1. Click the **Quick Filters** command in the Workflows panel.
2. Select one of the following choices to filter the items in the list:

Choice	Description
Show all workflows	Disables all previous filters and displays all workflows in process.
Hide/show completed workflows	Hides or shows workflows that have completed processing.
Hide/show terminated workflows	Hides or shows workflows that have been terminated.
Hide/show stopped workflows	Hides or shows workflows that have been stopped by user action.
Hide/show running workflows	Hides or shows workflows that are still running.

Sorting the List of Workflows

If you have a large number of request definitions, you might want to sort the list by a particular column, such as **Name** or **Description**.

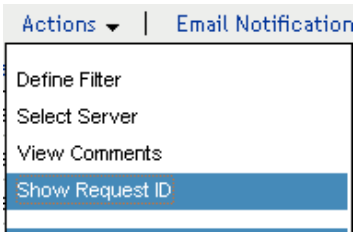
1 Click the heading for the sort column.

Displaying the Process Request ID

You can display and sort data based on the internal process ID for a request.

1 Click the **Actions** command in the Workflows panel.

2 Click **Show Request ID** on the **Actions** menu.



Depending on your display, you might need to scroll to the right to see the **Request ID** column.

To sort the data based on the process request ID, click the heading for the **Request ID** column.

Terminating a Workflow Instance

If you do not want a workflow instance to continue its processing, you can terminate the workflow.

1 Select the workflow in the Workflows panel by clicking the check box next to the workflow name.

2 Click the **Terminate** command in the Workflows panel.

Viewing Details about a Workflow Instance

When you have displayed a set of running workflows on a particular server, you can select a workflow instance to see more details about the running process.

NOTE: If a workflow instance uses a serial processing design pattern, the display shows a single activity as current because only one user can act on the work item at any point in time. However, if the workflow handles parallel processing and branching, there might be multiple current activities for a workflow instance.

To view details about a particular workflow instance:

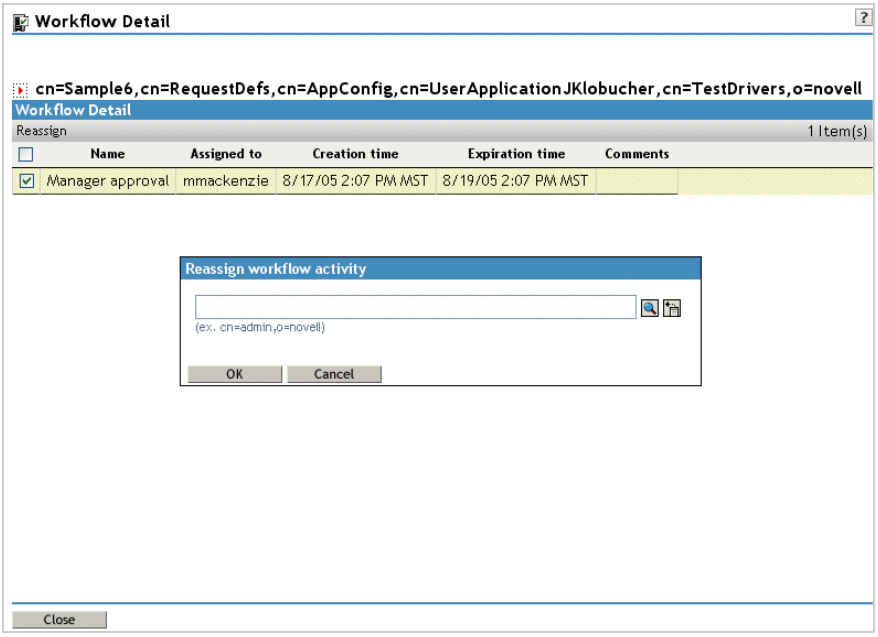
1 Click the name of the workflow instance in the Workflows panel.

iManager displays the Workflow Detail panel.

Reassigning a Workflow Instance

If a workflow instance has stopped and cannot be restarted, you can reassign the work item to another user or group.

1. Select the current activity associated with the workflow by clicking the check box next to the name in the Workflow Detail panel.
2. Click the **Reassign** command in the Workflow Detail panel.



3. Select the user or group to which you want to reassign the work item.

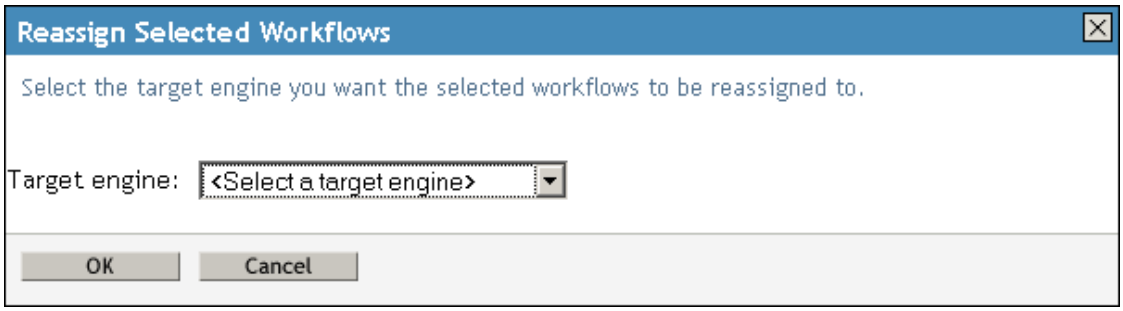
Managing Workflow Processes in a Cluster

You can use the Workflows screen to reassign processes from one workflow engine to another. For example, you could use this feature to reassign processes back to a failed workflow engine when the workflow engine is brought back online, or you could redistribute processes to other engines when an engine is permanently removed from the cluster.

The source engine(s) must be a in a SHUTDOWN or TIMEDOUT state. The target engine must be restarted in order to restart the processes that were reassigned to that engine.

Reassigning a Process from One Workflow Engine to Another

1. In the Workflows panel, select the workflow that you would like to reassign by clicking the check box next to the workflow name.
2. Select **Actions > Reassign**.



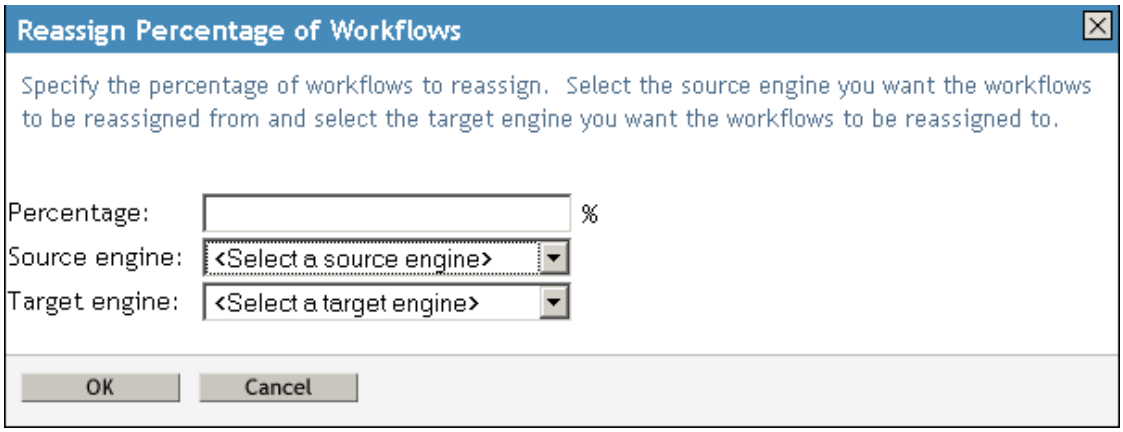
3 Select the workflow engine to which you want to reassign the workflow process from the **Target Engine** list.

4 Click **OK**.

Reassigning a Percentage of Processes from One Workflow Engine to Another

1 In the Workflows panel, select the workflow that you would like to reassign by clicking the check box next to the workflow name.

2 Select **Actions > Reassign Percentage**.



3 In the **Percentage** field, type the percentage of workflow processes that you would like to reassign from one workflow engine to another.

4 Use the **Source engine** list to select the workflow engine from which you want to reassign processes.

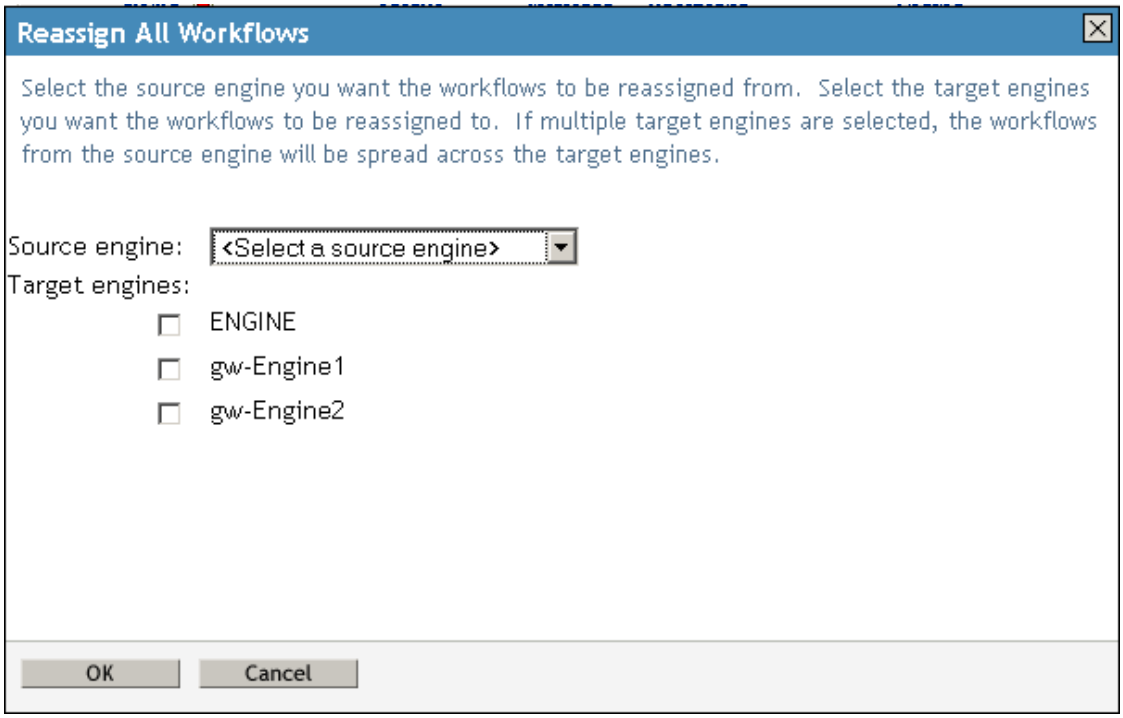
5 Use the **Target engine** field to select the workflow engine to which you want to reassign processes.

6 Click **OK**.

Reassigning All Processes from One Workflow Engine to Another

1 In the Workflows panel, select the workflow that you would like to reassign by clicking the check box next to the workflow name.

2 Select **Actions > Reassign All**.



3 Use the **Source engine** list to select the workflow engine from which you want to reassign processes.

4 Select the workflow engines to which you would like to reassign processes by clicking the check box next to the name of the workflow engine.

   If you select multiple target engines, the processes from the source engine will be evenly distributed to the target engine.

5 Click **OK**.

Configuring the Email Server

A workflow process often sends email notifications at various points in the course of its execution. For example, an email might be sent when a user assigns a workflow activity to a new addressee.

Before you can take advantage of the email notification capabilities of Identity Manager, you need to configure the SMTP email server. To do this, you need to use the **Email Server Options** task within the Workflow Administration role in iManager.

NOTE: This task is a shortcut to the **Email Server Options** task under the Passwords role.

To configure the email server:

1 Select the Identity Managercategory in iManager.

2 Open the **Workflow Administration** role.

3 Click on the **Email Server Options** task.

   iManager displays the Email Server Options panel.

4 Type the name (or IP address) of the host server in the **Host Name** field.

5 Type the email address for the sender in the **From** field.

   When the recipient opens the email, this text is displayed in the **From** field of the email header. Depending on your mail server settings, the text in this field might need to match a valid sender in the system in order to allow the mail server to do reverse lookups or authentication. An example is helpdesk@company.com instead of descriptive text such as The Password Administrator.

6 If your server requires authentication before sending email, select the **Authenticate to server using credentials** check box and specify the username and password.

7 When you are finished, click **OK**.

Working with Email Templates

Identity Manager includes email notification templates that are designed specifically for workflow-based provisioning. These email templates include the following.

- **New Provisioning Request** (Provisioning Notification)
- **Availability Setting Notification** (Availability)
- **Delegate Assignment Notification** (Delegate)
- **Provisioning Approval Notification** (Provisioning Approval Completed Notification)
- **Reminder - A request is waiting on your approval** (Provisioning Reminder)
- **Proxy Assignment Notification** (Proxy)
- **New Role Request** (Role Request Notification)
- **Role Request Approval Notification** (Role Request Approval Completed Notification)
- **Compliance Task** (Attestation Notification)
- **New Resource Request** (Resource Request Notification)
- **Resource Request Approval Notification** (Resource Request Approval Completed Notification)

The subject lines are listed first above. The template names (as they appear in iManager and Designer) are given in parentheses.

You can edit the templates to change the content and format of email messages. You can also create new templates. If you create new templates, you need to follow these naming conventions.

- The language-independent version of the Provisioning Notification template can have any name you like. The default template for notification email messages is called:

  ```
  Provisioning Notification
  ```

- The language-independent version of the Provisioning Reminder template can have any name you like. The default template for reminder email messages is called:

  ```
  Provisioning Reminder
  ```

- Each delegation template must have a name that begins with the word:

  ```
  delegate
  ```

  The language-independent name can be followed by one or more characters that describe the purpose or content of the template.

- Each proxy template must have a name that begins with the word:

`proxy`

The language-independent name can be followed by one or more characters that describe the purpose or content of the template.

- Each availability template must have a name that begins with the word:

  `availability`

  The language-independent name can be followed by one or more characters that describe the purpose or content of the template.

Each language-specific version of a template must have a suffix that provides a language code (for example, _fr for French, _es for Spanish, and so forth).

To create or edit an email template, use the **Email Templates** task within the Workflow Administration role in iManager.

NOTE: This task is a shortcut to the **Edit Email Templates** task under the Passwords role.

You also can create and edit email templates in Designer.

When you create a User Application driver in iManager or Designer, any email notification templates that are missing from the standard set of email notification templates are replaced. Existing email notification templates are not updated. This is to prevent overwriting email notification templates that you have customized. You can manually update the existing email notification templates using Designer. For more information, see About Email Notification Templates in the *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications* and Setting Up E-Mail Notification Templates in the *NetIQ Designer for Identity Manager Administration Guide*.

NOTE: When you use a localized email template in a provisioning request definition, the preferred locale setting of the recipient of the notification is ignored. For example, the Provisioning Notification of a request using a localized email notification template of Spanish will only send a Spanish email, regardless of the preferred locale setting for the user.

Default Content and Format

This section shows you what the content of the email templates looks like after you install the product. It also describes the replacement tags that can be used in the email template.

New Provisioning Request

This template identifies the provisioning request definition that triggered the email message. In addition, it includes a URL that redirects the addressee to the task that requires approval, as well as a URL that displays the complete list of tasks pending for that user.

```
Hi,

A new provisioning request has been submitted that requires your approval.

Request name: $requestTitle$
Submitted by: $initiatorFullName$
Recipient: $recipientFullName$

Please review the details of this request at $SECURE_PROTOCOL$://
$HOST$:$SECURE_PORT$/$TASK_DETAILS$ to take the appropriate action.

You can review a list of all requests pending your approval at
$SECURE_PROTOCOL$://$HOST$:$SECURE_PORT$/$TASKLIST_CONTEXT$.
```

Table 24-2 *New Provisioning Request Template: Replacement Tags*

Tag	Description
$userFirstName$	The first name of the addressee.
$requestTitle$	The display name of the provisioning request definition.
$initiatorFullName$	The full name of the initiator.
$recipientFullName$	The full name of the recipient.
$PROTOCOL$	The protocol for URLs included in the email message.
$SECURE_PROTOCOL$	The secure protocol for URLs included in the email message.
$HOST$	The host for the Tomcat application server that is running the identity applications. For information about setting the value for this parameter, see “Modifying Default Values for the Template” on page 253.
$PORT$	The port for the Identity Manager User Application. For information about setting the value for this parameter, see “Modifying Default Values for the Template” on page 253.
$SECURE_PORT$	The secure port for the Identity Manager User Application. For information about setting the value for this parameter, see “Modifying Default Values for the Template” on page 253.
$TASKLIST_CONTEXT$	The page that displays the list of all requests pending for the addressee.
$TASK_DETAILS$	The page that displays details for the request for which this email message was generated.

Availability Setting Notification

This template identifies a user whose availability has been updated. It includes the start time and expiration time of the period for which the user is unavailable, and the resources for which the user is unavailable.

```
Hi,

$submitterFirstName$ $submitterLastName$ has updated availability settings
for
  $userFirstName$ $userLastName$.
  This user has $operation$ an availability setting that applies to the
following resources:

  $resources$

  This setting indicates that $userFirstName$ $userLastName$ is unavailable
to work on these resources during the timeframe outlined below:

  Start time: $startTime$
  Expiration time: $expirationTime$

  When a user is unavailable, any delegates assigned may handle resource
requests for that user.

You can review a list of your availability settings at $SECURE_PROTOCOL$:/
/$HOST$:$SECURE_PORT$/$AVAILABILITY_CONTEXT$.
```

Table 24-3 *Availability Setting Notification Template: Replacement Tags*

Tag	Description
$submitterFirstName$	The first name of the user who updated the availability setting.
$PROTOCOL$	The protocol for URLs included in the email message.
$PORT$	The port for the Identity Manager User Application. For information about setting the value for this parameter, see "Modifying Default Values for the Template" on page 253.
$startTime$	The start time of the workflow for this provisioning request.
$resources$	The resources (provisioning requests) for which the addressee is unavailable.
$SECURE_PROTOCOL$	The secure protocol for URLs included in the email message.
$expirationTime$	The time at which the availability will expire.
$submitterLastName$	The last name of the user who updated the availability setting.
$SECURE_PORT$	The secure port for the Identity Manager User Application. For information about setting the value for this parameter, see "Modifying Default Values for the Template" on page 253.
$userFirstName$	The first name of the user to whom this availability setting applies.

Tag	Description
$userLastName$	The last name of the user to whom this availability setting applies.
$HOST$	The host for the Tomcat application server that is running the identity applications. For information about setting the value for this parameter, see “Modifying Default Values for the Template” on page 253.
$ASSIGNMENT_LIST_CONTEXT$	The context or path of the URL to the provisioning User Application.

Delegate Assignment Notification

This template notifies a user when a provisioning request has been submitted that requires the user’s approval. It includes the name of the request, the user who submitted the request, and the full name of the recipient. It includes links for viewing the provisioning request and for viewing all provisioning requests awaiting the user’s approval.

```
Hi,

A new provisioning request has been submitted that requires your approval.

Request name: $requestTitle$
Submitted by: $initiatorFullName$
Recipient: $recipientFullName$

Please review the details of this request at $SECURE_PROTOCOL$://
$HOST$:$SECURE_PORT$/$TASK_DETAILS$ to take the appropriate action.

You can review a list of all requests pending your approval at
$SECURE_PROTOCOL$://$HOST$:$SECURE_PORT$/$TASKLIST_CONTEXT$.
_SUBJECT
```

Table 24-4 *Delegate Assignment Notification: Replacement Tags*

Tag	Description
$submitterFirstName$	The first name of the user who assigned the delegate.
$PROTOCOL$	The protocol for URLs included in the email message.
$PORT$	The port for the Identity Manager User Application For information about setting the value for this parameter, see “Modifying Default Values for the Template” on page 253.
$resources$	The resources (provisioning requests) for which the delegate is available.
$SECURE_PROTOCOL$	The secure protocol for URLs included in the email message.

Tag	Description
$fromUsers$	The users for which the assigned delegate is authorized to handle resource requests.
$relationship$	The relationship defined in the directory abstraction layer that was selected for this delegate assignment.
$expirationTime$	The time at which the delegate assignment will expire.
$fromContainers$	The containers for which the assigned delegate is authorized to handle resource requests.
$fromGroups$	The groups for which the assigned delegate is authorized to handle resource requests.
$submitterLastName$	The last name of the user who assigned the delegate.
$SECURE_PORT$	The secure port for the Identity Manager User Application. For information about setting the value for this parameter, see "Modifying Default Values for the Template" on page 253.
$userFirstName$	The first name of the user who has been assigned as a delegate.
$userLastName$	The last name of the user who has been assigned as a delegate.
$HOST$	The host for the Tomcat application server that is running the identity applications. For information about setting the value for this parameter, see "Modifying Default Values for the Template" on page 253.
$ASSIGNMENT_LIST_CONTEXT$	The context or path of the URL to the provisioning User Application.

Provisioning Approval Notification

This template notifies a user when an approval process for a provisioning request submitted by the user has been completed.

```
Hi,

The approval process of your provisioning request has completed.

Request name: $requestTitle$
Request id: $requestId$
Submitted by: $initiatorFullName$
Submitted on: $requestSubmissionTime$
Recipient: $recipientFullName$

Status: $requestStatus$
```

Table 24-5 *Provisioning Approval Notification: Replacement Tags*

Tag	Description
$initiatorFullName$	The full name of the initiator.
$requestSubmissionTime$	The time at which the request was submitted.
$requestTitle$	The display name of the provisioning request definition.
$requestId	The ID of the provisioning request.
$recipientFullName$	The full name of the recipient.

Reminder - A Request Is Waiting on Your Approval

This template reminds a user that a provisioning request that requires the user's approval is waiting in a queue for approval. It includes the name of the request, the user who submitted the request, and the recipient. It includes links for viewing the provisioning request and for viewing all provisioning requests awaiting the user's approval.

```
Hi,

This is a reminder that a provisioning request is sitting in your queue
waiting on your approval.

Request name: $requestTitle$
Submitted by: $initiatorFullName$
Recipient: $recipientFullName$

Please review the details of this request at $SECURE_PROTOCOL$://
$HOST$:$SECURE_PORT$/$TASK_DETAILS$ to take the appropriate action.

You can review a list of all requests pending your approval at
$SECURE_PROTOCOL$://$HOST$:$SECURE_PORT$/$TASKLIST_CONTEXT$.
```

Table 24-6 *Reminder - A request is waiting on your approval: Replacement Tags*

Tag	Description
$TASKLIST_CONTEXT$	The page that displays the list of all requests pending for the addressee.
$PROTOCOL$	The protocol for URLs included in the email message.
$PORT$	The port for the Identity Manager User Application. For information about setting the value for this parameter, see "Modifying Default Values for the Template" on page 253.
$SECURE_PROTOCOL$	The secure protocol for URLs included in the email message.
$initiatorFullName$	The full name of the initiator.

Tag	Description
$recipientFullName$	The full name of the recipient.
$TASK_DETAILS$	The page that displays details for the request for which this email message was generated.
$SECURE_PORT$	The secure port for the Identity Manager User Application. For information about setting the value for this parameter, see "Modifying Default Values for the Template" on page 253.
$userFirstName$	The first name of the addressee.
$HOST$	The host for the Tomcat application server that is running the identity applications. For information about setting the value for this parameter, see "Modifying Default Values for the Template" on page 253.
$requestTitle$	The display name of the provisioning request definition.

Proxy Assignment Notification

This template notifies the recipient that a proxy has been assigned. The user who has been assigned as a proxy is identified, as are the users, groups, and containers for which the user is authorized to act as proxy. It includes links for viewing the recipient's list of proxy assignments.

```
Hi,

A proxy assignment that authorizes a user to act as proxy for
one or more users, groups, or containers was $operation$ by:
$submitterFirstName$ $submitterLastName$.
Unlike delegate assignments, proxy assignments are independent of resource
requests, and therefore apply to all work and settings actions.

The user selected as proxy is:

$userFirstName$  $userLastName$

The assigned proxy is authorized to handle all work for these users,
groups, and containers:

Users: $fromUsers$
Groups: $fromGroups$
Containers: $fromContainers$

This proxy assignment expires at:

$expirationTime$


You can review a list of your proxy assignments at $SECURE_PROTOCOL$://
$HOST$:$SECURE_PORT$/$PROXY_CONTEXT$.
```

Table 24-7 Proxy Assignment Notification: Replacement Tags

Tag	Description
$submitterFirstName$	The first name of the user who assigned the proxy.
$PROTOCOL$	The protocol for URLs included in the email message.
$PORT$	The port for the Identity Manager User Application. For information about setting the value for this parameter, see "Modifying Default Values for the Template" on page 253.
$resources$	The resources (provisioning requests) for which the proxy is available.
$SECURE_PROTOCOL$	The secure protocol for URLs included in the email message.
$fromUsers$	The users for which the assigned proxy is authorized to handle resource requests.
$expirationTime$	The time at which the proxy assignment will expire.
$fromContainers$	The containers for which the assigned proxy is authorized to handle resource requests.
$fromGroups$	The groups for which the assigned proxy is authorized to handle resource requests.
$submitterLastName$	The last name of the user who assigned the proxy.
$SECURE_PORT$	The secure port for the Identity Manager User Application. For information about setting the value for this parameter, see "Modifying Default Values for the Template" on page 253.
$userFirstName$	The first name of the user who has been assigned as a proxy.
$userLastName$	The last name of the user who has been assigned as a proxy.
$HOST$	The host for the Tomcat application server that is running the identity applications. For information about setting the value for this parameter, see "Modifying Default Values for the Template" on page 253.
$ASSIGNMENT_LIST_CONTEXT$	The context or path of the URL to the provisioning User Application.

New Role Request

This template identifies the provisioning request definition that triggered the email message. In addition, it includes a URL that redirects the addressee to the task that requires approval, as well as a URL that displays the complete list of tasks pending for that user.

```
Hi,

A new role request has been submitted that requires your approval.

Request name: $requestTitle$
Submitted by: $initiatorFullName$
Recipient: $recipientFullName$

Please review the details of this role request at $SECURE_PROTOCOL$://
$HOST$:$SECURE_PORT$/$TASK_DETAILS$ to take the appropriate action.

You can review a list of all role requests pending your approval at
$SECURE_PROTOCOL$://$HOST$:$SECURE_PORT$/$TASKLIST_CONTEXT$.
```

Table 24-8 *New Role Request Template: Replacement Tags*

Tag	Description
$userFirstName$	The first name of the addressee.
$requestTitle$	The display name of the request definition.
$initiatorFullName$	The full name of the initiator.
$recipientFullName$	The full name of the recipient.
$PROTOCOL$	The protocol for URLs included in the email message.
$SECURE_PROTOCOL$	The secure protocol for URLs included in the email message.
$HOST$	The host for the Tomcat application server that is running the identity applications. For information about setting the value for this parameter, see “Modifying Default Values for the Template” on page 253.
$PORT$	The port for the Identity Manager User Application. For information about setting the value for this parameter, see “Modifying Default Values for the Template” on page 253.
$SECURE_PORT$	The secure port for the Identity Manager User Application. For information about setting the value for this parameter, see “Modifying Default Values for the Template” on page 253.
$TASKLIST_CONTEXT$	The page that displays the list of all requests pending for the addressee.
$TASK_DETAILS$	The page that displays details for the request for which this email message was generated.

Role Request Approval Notification

This template notifies a user when an approval process for a role request submitted by the user has been completed.

```
Hi,

The approval process of your role request has completed.

Request name: $requestTitle$
Request id: $requestId$
Submitted by: $initiatorFullName$
Submitted on: $requestSubmissionTime$
Recipient: $recipientFullName$

Status: $requestStatus$
```

Table 24-9 *Role Request Approval Notification: Replacement Tags*

Tag	Description
$initiatorFullName$	The full name of the initiator.
$requestSubmissionTime$	The time at which the request was submitted.
$requestTitle$	The display name of the provisioning request definition.
$requestId	The ID of the role request.
$recipientFullName$	The full name of the recipient.

Compliance Task

This template notifies an attester when an attestation process has assigned a task to the attester.

```
Hi,

A new compliance activity has been submitted that requires your attention.

Request name: $requestTitle$
Submitted by: $initiatorFullName$

Please review the details of this compliance activity request at
$SECURE_PROTOCOL$://$HOST$:$PORT$/$TASK_DETAILS$ to take the appropriate
action.

You can review a list of all requests pending your action at
$SECURE_PROTOCOL$://$HOST$:$SECURE_PORT$/$TASKLIST_CONTEXT$.
```

Table 24-10 *Compliance Task: Replacement Tags*

Tag	Description
$initiatorFullName$	The full name of the initiator.
$requestTitle$	The display name of the attestation request.

New Resource Request

This template identifies the resource request definition that triggered the email message. In addition, it includes a URL that redirects the addressee to the task that requires approval, as well as a URL that displays the complete list of tasks pending for that user.

```
Hi,

A new resource request has been submitted that requires your approval.

Request name: $requestTitle$
Submitted by: $initiatorFullName$
Recipient: $recipientFullName$

Please review the details of this role request at $SECURE_PROTOCOL$://
$HOST$:$PORT$/$TASK_DETAILS$ to take the appropriate action.

You can review a list of all resource requests pending your approval at
$SECURE_PROTOCOL$://$HOST$:$SECURE_PORT$/$TASKLIST_CONTEXT$.
```

Table 24-11 *New Resource Request Template: Replacement Tags*

Tag	Description
$userFirstName$	The first name of the addressee.
$requestTitle$	The display name of the request definition.
$initiatorFullName$	The full name of the initiator.
$recipientFullName$	The full name of the recipient.
$PROTOCOL$	The protocol for URLs included in the email message.
$SECURE_PROTOCOL$	The secure protocol for URLs included in the email message.
$HOST$	The host for the Tomcat application server that is running the identity applications.
$PORT$	The port for the Identity Manager User Application.
$SECURE_PORT$	The secure port for the Identity Manager User Application.
$TASKLIST_CONTEXT$	The page that displays the list of all requests pending for the addressee.
$TASK_DETAILS$	The page that displays details for the request for which this email message was generated.

Resource Request Approval Notification

This template notifies a user when an approval process for a resource request submitted by the user has been completed.

```
Hi,

The approval process of your resource request has completed.

Request name: $requestTitle$
Request id: $requestId$
Submitted by: $initiatorFullName$
Submitted on: $requestSubmissionTime$
Recipient: $recipientFullName$

Status: $requestStatus$
```

Table 24-12 *Role Request Approval Notification: Replacement Tags*

Tag	Description
`$initiatorFullName$`	The full name of the initiator.
`$requestSubmissionTime$`	The time at which the request was submitted.
`$requestTitle$`	The display name of the provisioning request definition.
`$requestId`	The ID of the role request.
`$recipientFullName$`	The full name of the recipient.

Editing Email Templates

You can change the content or format of the supplied email templates. For information about creating email templates, see "Configuring Email Notification" in the *NetIQ Identity Manager Administration Guide.*

To edit a template:

1. Select the **Identity Manager** category in iManager.
2. Open the **Workflow Administration** role.
3. Click the **Email Templates** task.

   iManager displays the **Edit Email Templates** panel.
4. Click the name of the email template that you would like to edit.

   iManager displays the **Modify Email Message** screen.
5. Make your changes in the **Message Body** box.
6. If necessary, copy one or more of the supplied tags in the **Replacement Tags** list to include dynamic text in the message body.

   For a description of the replacement tags, see "Default Content and Format" on page 241.
7. When you are finished, click **OK**.

Modifying Default Values for the Template

At installation time, you can set default values for several of the replacement tags used in email templates. After you have completed the installation, you can also modify these values by using the Configuration Update utility.

1 Run the `configupdate.sh` script in the `idm` folder.

```
./configupdate.sh
```

On Windows, run `configupdate.bat`.

2 Make changes as necessary to any of the following fields:

Field	Description
Email Notify Host	Used to replace the $HOST$token in email templates used in approval flows. If left blank, computed by the server.
Email Notify Port	Used to replace the $PORT$token in email templates used in approval flows.
Email Notify Secure Port	Used to replace the $SECURE_PORT$token in email templates used in approval flows.

3 Click **OK** to confirm your changes.

Adding Localized Email Templates

To add localized email templates:

1 Select the **Identity Manager** category in iManager.

2 Open the **Workflow Administration** role.

3 Click the **Email Templates** task.

iManager displays the **Edit Email Templates** panel.

4 Identify the email template (without any locale in the name) that you want to copy.

4a Write down the template name to use in Step 5.

4b Click the template subject to open the template and view its message subject, body, and replacement tags.

4c Copy the message subject, body (to be translated), and replacement tags that you want to use in your new template.

4d Click **Cancel**.

5 Click **Create**, then enter the template name with a locale extension. For example, to create a Forgot Hint template in German, enter the name Forgot Hint_de, where _de signifies Deutsch (German).

If you use a two-letter language and two-letter country code, this works fine. If you attempt to use a locale with a variant such as en_US_TX, only the variant and language are considered. Do not use locale variants when naming email templates.

6 Click **OK**.

7 In the template list, click the newly created template, for example Forgot Hint_de, and enter the translated subject and message body. Be sure to preserve the replacement tags surrounded by the dollar ($) sign in the message body.

8 If necessary, copy one or more of the supplied tags in the **Replacement Tags** list to include dynamic text in the message body.

For a description of the replacement tags, see "Default Content and Format" on page 241.

9 Click **Apply**.

10 Click **OK**.

NOTE: Email templates only send localized content if the preferred locale is set for the user (to whom the mail is sent).

Allowing a Named Password to be Retrieved over LDAP

You can add a boolean definition to the User Application driver to allow a named password to be retrieved over LDAP from a workflow. To take advantage of this feature, you need to create a global configuration value "allow-fetch-named-passwords".

Here's a sample definition:

```
<definitions>
    <definition display-name="Allow Named Password to be retrieved over
LDAP"
name="allow-fetch-named-passwords" type="boolean">
        <value>false</value>
        <description>Allow Named Password to be retrieved over LDAP. If the
value is true, then the named password value can be fetched using the LDAP
extension
com.novell.nds.dirxml.ldap.GetNamedPasswordRequest/
com.novell.nds.dirxml.ldap.GetNamedPasswordResponse.</description>
    </definition>
</definitions>
```

If the global configuration is not present, the runtime functions as if the definition is present and the value is set to false. If you then try to use the GCV script method getValueForNamedPassword(String valueKey), an exception is thrown since the permission is set to false. If you want to be able to use the method, then the value for allow-fetch-named-passwords variable must be true.

If the gcv variable allow-fetch-named-passwords does not exist, you have to create the variable and set it to `true`. If it already exists, you can simply need to set the value to true.

NOTE: To retrieve a named password, you must use the GCV script method getValueForNamedPassword on a GCV of the `password-ref` type, which points to the named password. You cannot use the get script method.

To add the GCV value for the `allow-fetch-named-passwords` option:

1 In iManager, double click on the User Application driver.

2 Click on the Global Configuration Values tab.

3 Click on the *Add* button.

4 Fill out the definition, as described below:

   4a Specify `allow-fetch-named-passwords` as the name for the global configuration definition.

   4b Specify `Allow Named Password to be retrieved over LDAP` as the display name.

   4c Provide a description for the definition.

   4d Specify **boolean** as the Type.

5 Click *OK*.

6 Set the value to true or false and click *Apply*.

7 Create a named password in your User Application driver.

8 Create a GCV of the type `password-ref` that points to the named password you want to be able to read.

9 In your workflow, use the function getValueForNamedPassword to retrieve the value of the named password, using the following syntax:

```
GCV.getValueForNamedPassword('PasswordRefGCV')
```

V Web Service Reference

These sections describe the Web Service endpoints provided for the User Application.

- Chapter 25, “Provisioning Web Service,” on page 259
- Chapter 26, “Metrics Web Service,” on page 331
- Chapter 27, “Notification Web Service,” on page 351
- Chapter 28, “Directory Abstraction Layer (VDX) Web Service,” on page 361
- Chapter 29, “Role Web Service,” on page 387
- Chapter 30, “Resource Web Service,” on page 487
- Chapter 31, “Forgot Password Web Service,” on page 523

25 Provisioning Web Service

This section describes the Provisioning Web Service, which allows SOAP clients to access Provisioning functionality.

About the Provisioning Web Service

The Identity Manager User Application includes a workflow system that executes approval flows. A workflow process is based on a provisioning request definition, which is an XML document stored in the Identity Vault. The provisioning request definition describes an arbitrary topology using activities and links. For example, a provisioning request to grant an entitlement might have a workflow that collects approvals from relevant users and writes the entitlement to the directory.

To support access by third-party software applications, the provisioning workflow system includes a Web service endpoint. The endpoint offers all provisioning functionality (for example, allowing SOAP clients to start a new approval flow, or list currently executing flows). The Web service is built using the NetIQ Web Service SDK (WSSDK), which supports the WS-I Basic Profile, thus guaranteeing interoperability with other standards based SOAP implementations.

This Appendix describes the provisioning Web service in detail and shows how to access it using the Web or by writing a Java or C# client. We provide an overview of the operations in the SOAP endpoint and describe how to use the Web interface. We show how to develop a Java client using the SOAP toolkit included with Identity Manager provisioning, followed by how to write a C# client using Mono. The sample source code a the Java client and associated ANT build file is provided.

Provisioning Web Service Overview

Identity Manager is composed of two main systems: the Identity Vault and the workflow application. The Identity Vault is capable of connecting to a large number of different systems such as databases, financial systems, and other enterprise applications, and keep these systems synchronized. The rules for synchronizing the remote systems can be very complex and the Identity Vault engine supports a sophisticated scripting language for expressing the rules.

The workflow application is composed of several subsystems. The User Application provides a user-interface for workflows. The User Application is a Web application for requesting and managing approval flows. The Web application runs in a portal, which also includes administration portlets. The workflow application contains a security layer, a directory abstraction layer and a logging subsystem, which can send log events to NetIQ Sentinel. The workflow subsystem is responsible for executing approval flows. The User Application runs on a Tomcat application server and uses a database (for example, Oracle) for persistence.

The Web service for the workflow system is only used by the User Application driver, which is capable of listening to certain events emitted by the Identity Vault engine and convert these events into an appropriate SOAP message. For example, when a specific attribute in the Identity Vault changes, the Identity Vault engine emits an event, which the User Application picks up from the subscriber channel. The User Application driver then sends a SOAP message to the provisioning Web service to start a new approval flow.

Removing Administrator Credential Restrictions

By default, the requirement for invoking the public interfaces for the SOAP services is that the HTTP session logged in user must have administrator credentials. The Provisioning and Directory Services require Provisioning Administrator credentials. The Role Service and Resource Service require Role Administrator and Resource Administrator credentials respectively. The restrictions can be removed to allow a session with a logged in user who does not have administrator credentials to invoke the methods for the services by changing the configuration settings for the service. The details for changing the Provisioning Service follow. Instructions for the other SOAP services are provided with the documentation for these services.

To remove the administrator credential restriction for the Provisioning Service:

1. Open the `ism-configuration.properties` file, located by default in the `/netiq/idm/apps/tomcat/conf` directory.
2. Change `WorkflowService/SOAP-End-Points-Accessible-By-ProvisioningAdminOnly` to `false`.
3. Save and close the file.

These are the methods that can be invoked by users without Provisioning Administrator credentials if the WorkflowService/SOAP-End-Points-Accessible-By-ProvisioningAdminOnly property is set to false:

- getAllProvisioningRequests(String)
- getDataItems(String workId)
- getDefinitionByID(String definitionID, String recipient)
- getProvisioningCategories()
- getProvisioningRequests(String recipient, String category, String operation)
- getWork(String workId)
- getWorkEntries(T_WorkEntryQuery query, int maxRecords)
- start(String processId, String recipient, DataItemArray items)
- startAsProxy(String processId, String recipient, DataItemArray items, String proxyUser)
- startAsProxyWithDigitalSignature(String processId, String recipient, DataItemArray items, String digitalSignature, SignaturePropertyArray digitalSignaturePropertyArray, String proxyUser)
- startWithCorrelationId(String processId, String recipient, DataItemArray items, String digitalSignature, SignaturePropertyArray digitalSignaturePropertyArray, String proxyUser, String correlationId)
- startWithDigitalSignature(String processId, String recipient, DataItemArray items, String digitalSignature, SignaturePropertyArray digitalSignaturePropertyArray)

All other methods for this service always require Provisioning Administrator credentials independent of whether the WorkflowService/SOAP-End-Points-Accessible-By-ProvisioningAdminOnly property is set to false.

Provisioning Web Service Method Categories

The methods provided by the provisioning Web service endpoint are divided into six categories:

Table 25-1 *Provisioning Web Service Operation Categories*

Category	Description
Comments	Methods for retrieving comments and for adding a comment to a pending user activity
Configuration	Methods for getting and setting configuration parameters for the workflow system (for example, timeouts, thread pool settings).
Miscellaneous	Several unrelated methods (for example, for getting a JPG with a provisioning request's topology, for getting the XML definition of a provisioning request, and for getting the XML for the request form).
Processes	Methods for getting information about running and completed workflow processes.
Provisioning Requests	Methods for working with provisioning requests (for example, listing available provisioning requests, listing provisioning categories)
Work Entries	Methods for retrieving and manipulating work entries (items awaiting approval).

The methods provided by the provisioning Web service are described in detail in Section 25, “Provisioning Web Service,” on page 259.

Developing Clients for the Provisioning Web Service

Web Access to the Provisioning Web Service

A SOAP-based Web service is usually accessed by inserting a SOAP message in the body of an HTTP Post request.The Web service toolkit used to build the provisioning Web service also supports access using HTTP GET. In other words, you can open the URL of the Web service endpoint in a browser and interact with the Web service. In particular, the provisioning Web service lets you invoke each of its operations.

Accessing the Test Page

You can access the provisioning Web Service endpoint using a URL similar to the following:

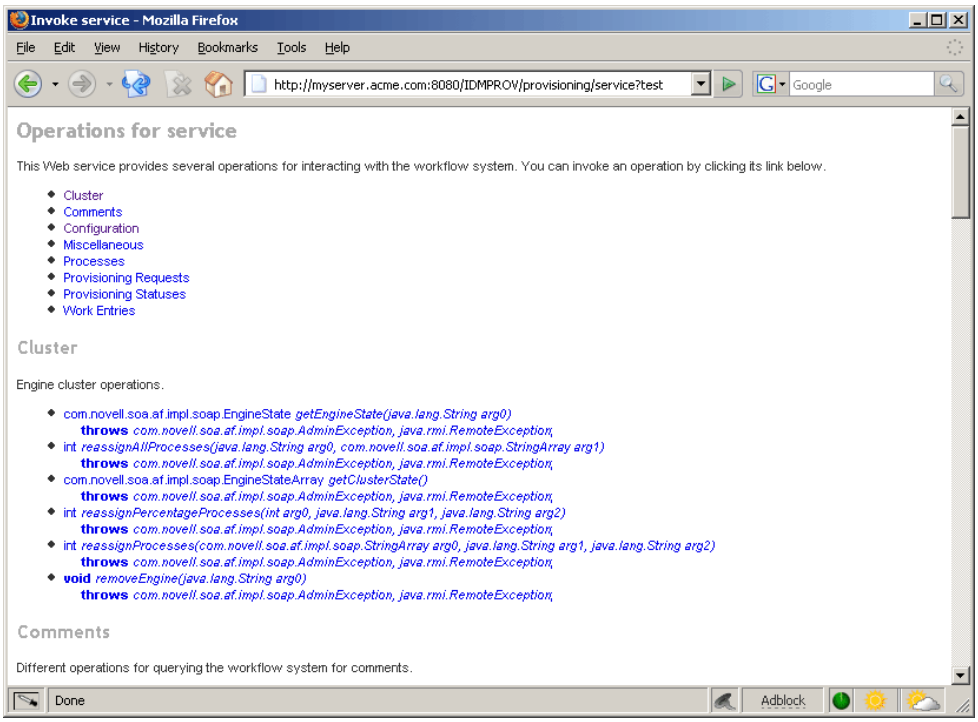
`http://`*`server`*`:`*`port/warcontext`*`/provisioning/service?test`

For example, if your server is named “myserver”, your User Application is listening on port 8080, and your User Application war file is named “IDMPROV”, the URL would be:

`http://myserver:8080/IDMPROV/provisioning/service?test`

The following page is displayed:

Figure 25-1 *Web Service Test Page*



You can also access the SOAP endpoint by going to the **Administration** within the User Application. To do this, you need to select the **Application Configuration** tab, then select **Web Services** from the left-navigation menu. After selecting **Web Services**, pick the Web Service endpoint you want from the list.

WARNING: The test page is disabled by default. Since some of the methods allow data to be updated, the test page presents a potential security vulnerability and should not be allowed in a production environment. For details on enabling the test page, see the instructions provided for the Role Service in "Accessing the Test Page" on page 261.

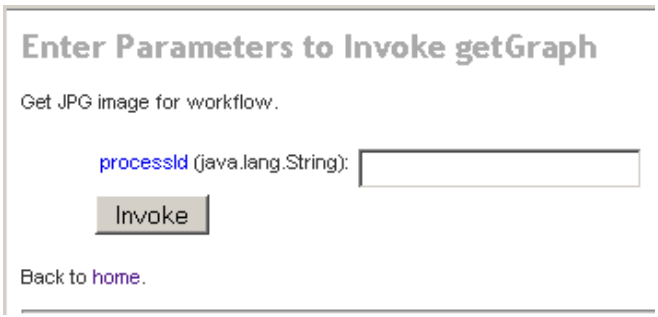
Entering Arguments for Operations

To see an example of an operation that is particularly useful to invoke from the browser, scroll down to the **Miscellaneous** section and click **getGraph**.

NOTE: The Graphviz program must be installed on the computer where the application server and the IDM User Application is running. For more information about Graphviz, see Graphviz (http://www.graphviz.org).

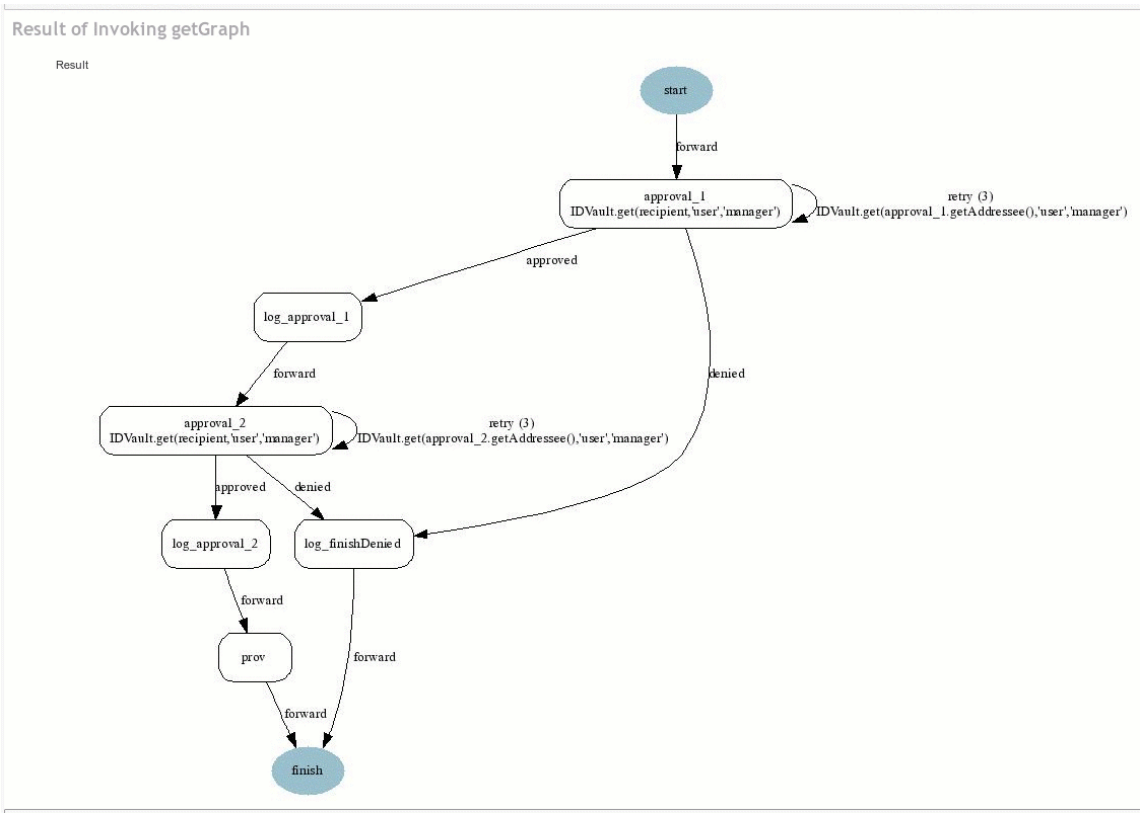
A page is displayed that allows you to enter the parameters for the getGraph method.

Figure 25-2 *Parameters for getGraph Method*



The method takes one argument, which is the distinguished name of a provisioning request. Enter the DN, and the underlying workflow is displayed as a JPG file..

Figure 25-3 *Output of getGraph*



A Java Client for the Provisioning Web Service

This section describes how to develop a simple Java client for the provisioning Web service, which lists all the processes in the workflow system. For complete source code for the client, see “Sample Code for the Java Client” on page 267.

Prerequisites

To develop a Java client you must install a supported Java Developer's Kit. Also, a client program needs the following JAR files:

activation.jar
commons-httpclient.jar
IDMfw.jar
IDMrbac.jar
log4j.jar
saaj-api.jar
wssdk.jar
commons-codec-1.3.jar
commons-logging.jar
jaxrpc-api.jar
mail.jar
workflow.jar
xpp3.jar

Developing a Java Client

Developing a client that accesses a Web service consists of two steps:

- Get the stub, which is the object that represents the remote service
- Invoke one or more of the operations available in the remote service

The Java programming model for Web services is very similar to RMI. The first step is to lookup the stub using JNDI:

```
InitialContext ctx = new InitialContext();
ProvisioningService service = (ProvisioningService)
ctx.lookup("xmlrpc:soap:com.novell.soa.af.impl.soap.ProvisioningService");
Provisioning prov = service.getProvisioningPort();
```

The first line of code creates the initial context for JNDI lookups. The second line looks up the service object, which is a kind of factory that can be used to retrieve the stub for the provisioning Web service. The last line gets the provisioning stub from the service.

Before invoking an operation on the provisioning stub, it is necessary to set some properties, including the credentials used for authentication on the service, as well as the endpoint URL.

```
Stub stub = (Stub) prov;
// set username and password
stub._setProperty(Stub.USERNAME_PROPERTY, USERNAME);
stub._setProperty(Stub.PASSWORD_PROPERTY, PASSWORD);
// set the endpoint URL
stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, url);
```

These and other stub properties are described in more detail in "Frequently Used Stub Constants" on page 265. Now that we have a fully configured stub, we can invoke the getAllProcesses operation and dump information about each of the processes returned on the console:

```
// invoke the getAllProcesses method
ProcessArray array = prov.getAllProcesses();
Process[] procs = array.getProcess();
// print process array
System.out.println("list of all processes:");
if (procs != null) {
for (int i = 0; i < procs.length; i++) {
System.out.println(" process with request identifier " +
procs[i].getRequestId());
System.out.println(" initiator = " + procs[i].getInitiator());
System.out.println(" recipient = " + procs[i].getRecipient());
System.out.println(" processId = " + procs[i].getProcessId());
procs[i].getCreationTime().getTime());
if (null != procs[i].getCompletionTime()) {
System.out.println(" completed = " +
procs[i].getCompletionTime().getTime());
}
System.out.println(" approval status = " +
procs[i].getApprovalStatus());
System.out.println(" process status = " +
procs[i].getProcessStatus());
if (i != procs.length - 1)
System.out.println();
}
}
```

A method invocation on the stub results in a SOAP message being sent using the HTTP transport to the provisioning Web service. For operations that have arguments, the stub takes care of marshaling those Java objects into XML. The Web service returns a SOAP message, and the stub unmarshals the XML, in this case converting it into a ProcessArray Java object.

Running the Client

The sample ANT build file has a target for running the client (see “Sample Ant File” on page 269). The client needs the JAR files described in “Prerequisites” on page 264 to be in the CLASSPATH. You can change the code to have a different default address for the provisioning Web service SOAP endpoint, or simply specify it as a command line argument. For example:

```
ant -Durl=http://www.company.com:80/IDMProv/provisioning/service run
```

Frequently Used Stub Constants

The com.novell.soa.ws.portable.Stub class (which is part of WSSDK) supports several properties that can be used to configure a stub instance (for example, to fine-tune aspects of the HTTP communication). The following table lists a small subset of these properties, which are frequently used:

Table 25-2 *Provisioning Web Service Stub Constants*

Property	Type	Description
ENDPOINT_ADDRESS_PROPERTY	java.lang.String	The URL of the Web service. The URL protocol scheme can be HTTP or HTTPS depending on the requirements of the server. The path portion should be: `/IDMProv/provisioning/service`
HTTP_HEADERS	java.util.Map	Additional HTTP headers as String name/value pairs.
HTTP_TIME_OUT	java.lang.Integer	The number of milliseconds to wait to establish a connection to the host before timing out.
HTTP_MAX_TOTAL_CONNECTIONS	java.lang.Integer	The number of concurrent connections that this client program can establish to all server hosts it accesses. The default limit is 20.
HTTP_MAX_HOST_CONNECTIONS	java.lang.Integer	The number of concurrent connections this client program can establish to an individual server host. The default limit is 2. This value may not exceed that of `HTTP_MAX_TOTAL_CONNECTIONS`, so if a client requires more than 20 connections to the server, it must also set `HTTP_MAX_TOTAL_CONNECTIONS` to the desired value.
USERNAME	java.lang.String	The user ID for HTTP authentication.
PASSWORD	java.lang.String	The password for HTTP authentication.
HTTP_PROXY_HOST	java.lang.String	The host DNS name of a proxy. Setting this property requires setting HTTP_PROXY_PORT as well.
HTTP_PROXY_PORT	java.lang.Integer	The port to use on a proxy. Setting this property requires setting HTTP_PROXY_HOST as well.
HTTP_PROXY_AUTH_SCHEME	java.lang.Integer	The authentication scheme (Basic or Digest) to use for a proxy.
HTTP_PROXY_USERNAME	java.lang.String	The user ID for HTTP authentication using a proxy.
HTTP_PROXY_PASSWORD	java.lang.String	The password for HTTP authentication via proxy.

The TCP Tunnel

The TCP Tunnel is a useful tool for looking at the SOAP messages that are exchanged between a client and a server. The ANT build file (see “Sample Ant File” on page 269) has a target for starting the tunnel. Once the tunnel starts you need to enter the port on which the tunnel will listen, and the host/port of the remote Web service. The default settings cause the tunnel to listen on port 9999 and connect to a service running on localhost port 8080. The client program (see “Developing a Java

Client" on page 264) uses the first command line parameter to set the ENDPOINT_ADDRESS_PROPERTY. Using the default values, you can run the client using the following command, after starting the tunnel:

```
ant -Durl=http://localhost:9999/IDMProv/provisioning/service run
```

Sample Code for the Java Client

The following is the code for the Java client for listing all processes in the workflow system

```
package com.novell.examples;
import javax.naming.InitialContext;
import com.novell.soa.af.impl.soap.AdminException;
import com.novell.soa.af.impl.soap.Process;
import com.novell.soa.af.impl.soap.ProcessArray;
import com.novell.soa.af.impl.soap.Provisioning;
import com.novell.soa.af.impl.soap.ProvisioningService;
import com.novell.soa.ws.portable.Stub;
public class Client
{
private static final String USERNAME = "admin";
private static final String PASSWORD = "test";
public static void main(String[] args)
{
try {
String url = args.length > 0 ? args[0] :
"http://localhost:8080/IDMProv/provisioning/service";
listProcesses(url);
} catch (AdminException ex) {
System.out.println("command failed: " + ex.getReason());
} catch (Exception ex) {
ex.printStackTrace();
}
}
private static void listProcesses(String url)
throws Exception
{
// get the stub
InitialContext ctx = new InitialContext();
ProvisioningService service = (ProvisioningService)
ctx.lookup("xmlrpc:soap:com.novell.soa.af.impl.soap.ProvisioningService");
Provisioning prov = service.getProvisioningPort();
Stub stub = (Stub) prov;
// set username and password
stub._setProperty(Stub.USERNAME_PROPERTY, USERNAME);
stub._setProperty(Stub.PASSWORD_PROPERTY, PASSWORD);
// set the endpoint URL
stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, url);
// invoke the getAllProcesses method
ProcessArray array = prov.getAllProcesses();
Process[] procs = array.getProcess();
// print process array
System.out.println("list of all processes:");
if (procs != null) {
for (int i = 0; i < procs.length; i++) {
```

```
System.out.println(" process with request identifier " +
procs[i].getRequestId());
System.out.println(" initiator = " + procs[i].getInitiator());
System.out.println(" recipient = " + procs[i].getRecipient());
System.out.println(" processId = " + procs[i].getProcessId());
System.out.println(" created = " +
procs[i].getCreationTime().getTime());
if (null != procs[i].getCompletionTime()) {
System.out.println(" completed = " +
procs[i].getCompletionTime().getTime());
}
System.out.println(" approval status = " +
procs[i].getApprovalStatus());
System.out.println(" process status = " +
procs[i].getProcessStatus());
if (i != procs.length - 1)
System.out.println();
}
}
}
}
```

Developing a Mono Client

The previous section described how to create a Java client using the Web service toolkit and the pre-compiled stub code included with Identity Manager. This section describes how to develop a client using just the WSDL for the provisioning Web service. This example uses Mono and creates a C# client that changes the default retention time of 120 days for completed workflows to 30.

Prerequisites

To get started, you need to download Mono and install it on your system (see the Mono Project Website (http://www.mono-project.com/)). The version of Mono available at the time this document was written did not support complex schema types in which an element has the nillable attribute set to true. Because this construct is used in the provisioning WSDL, you must manually edit the Provisioning.WSDL file and remove the three places where `nillable="true"` is used.

Generating the Stub

Compared to the Java client developed in “Developing a Java Client” on page 264, there is one additional step required when building the C# client. Since the stub for accessing the Web service SOAP endpoint is not provided, you must generate the stub from the WSDL document. Mono includes a compiler called `wsdl` that processes the WSDL file and creates the stub. You can download the WSDL file from your User Application server by accessing the following URL:

`http://`*`myserver`*`:8080/`*`IDMProv`*`/provisioning/service?wsdl`

Replace “myserver” with the name of your server, and “IDMProv” with the name of your User Application war file.

Compile the WSDL file using the following command:

```
wsdl Provisioning.wsdl
```

This will generate a C# file called ProvisioningService.cs, which you need to compile into a DLL using the following Mono C# compiler command:

```
mcs /target:library /r:System.Web.Services.dll ProvisioningService.cs
```

Compared to the Java client, the resulting `ProvisioningService.dll` file is the equivalent of `workflow.jar`, which contains the stub code and supporting classes for accessing the provisioning Web service. The following is the source code for the simple C# client that sets the flow retention time and displays the new value on the console:

```
using System;
using System.Net;
class provclient {
public static void Main(string [] args) {
// create the provisioning service proxy
ProvisioningService service = new ProvisioningService();
// set the credentials for basic authentication
service.Credentials = new NetworkCredential("admin", "test");
service.PreAuthenticate = true;
// set the value for completed request retention to 30 days
setCompletedProcessTimeoutRequest req = new
setCompletedProcessTimeoutRequest();
req.arg0 = 30;
service.setCompletedProcessTimeout(req);
// display the new value on the console
getCompletedProcessTimeoutResponse res =
service.getCompletedProcessTimeout(new
getCompletedProcessTimeoutRequest());
Console.WriteLine(res.result);
}
}
```

You need to edit the file using the administrator credentials on your deployed Identity Manager system. Compile the client using the following command:

```
mcs /r:ProvisioningService.dll /r:System.Web provclient.cs
```

This generates the `provclient.exe` file.

Running the Client

Use the following command to run the client:

```
mono provclient.exe
```

Sample Ant File

The sample Ant file includes useful targets for extracting the necessary JAR files from the Identity Manager installation, compiling and running the Java client, and for launching the TCP Tunnel.

```
<?xml version="1.0"?>
<project name="client" default="all" basedir=".">
<target name="all" depends="clean, extract, compile"></target>
<!-- main clean target -->
<target name="clean">
<delete quiet="true" dir="classes"/>
<delete quiet="true" dir="lib"/>
</target>
<!-- init sets up the build environment -->
<target name="init">
<mkdir dir="classes"/>
<copy todir="${basedir}/lib">
<fileset dir="${basedir}" includes="log4j.properties"/>
</copy>
<!-- classpath -->
<path id="CLASSPATH">
<pathelement location="${basedir}/classes"/>
<fileset dir="${basedir}/lib" includes="*.jar"/>
</path>
</target>
<!-- extract -->
<target name="extract">
<property name="idm.home" value="/opt/netiq/idm3"/>
<property name="tomcat.lib" value="${idm.home}/tomcat-4.0.3/server/
IDMProv/lib"/>
<mkdir dir="lib"/>
<unzip src="${idm.home}/IDMProv.war" dest="${basedir}/lib">
<patternset>
<include name="WEB-INF/lib/commons-codec-1.3.jar"/>
<include name="WEB-INF/lib/commons-httpclient.jar"/>
<include name="WEB-INF/lib/commons-logging.jar"/>
<include name="WEB-INF/lib/jaxrpc-api.jar"/>
<include name="WEB-INF/lib/saaj-api.jar"/>
<include name="WEB-INF/lib/xpp3.jar"/>
<include name="WEB-INF/lib/workflow.jar"/>
<include name="WEB-INF/lib/wssdk.jar"/>
<include name="WEB-INF/lib/IDMfw.jar"/>
</patternset>
</unzip>
<move todir="${basedir}/lib">
<fileset dir="${basedir}/lib/WEB-INF/lib" includes="*.jar"/>
</move>
<delete quiet="true" dir="${basedir}/lib/WEB-INF"/>
<copy todir="${basedir}/lib">
<fileset dir="${tomcat.lib}" includes="activation.jar, mail.jar,
log4j.jar"/>
</copy>
</target>
<!-- tunnel -->
<target name="tunnel" depends="init">
<java classname="com.novell.soa.ws.impl.tools.tcptunnel.Tunnel"
fork="true"
spawn="true">
<classpath refid="CLASSPATH"/>
```

```
</java>
</target>
<!-- compile -->
<target name="compile" depends="init">
<javac srcdir="${basedir}" destdir="classes"
includes="Client.java">
<classpath refid="CLASSPATH"/>
</javac>
</target>
<!-- run -->
<target name="run" depends="init">
<property name="url" value="http://localhost:8080/IDMProv/provisioning/
service"/>
<java classname="com.novell.examples.Client" fork="true">
<arg line="${url}"/>
<classpath refid="CLASSPATH"/>
</java>
</target>
</project>
```

Sample Log4J File

The following log4j file sets the default log level to “error”:

```
log4j.rootCategory=ERROR, R
log4j.appender.R=org.apache.log4j.ConsoleAppender
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%-5p: %m%n
```

Provisioning Web Service API

This section provides details about the Provisioning Web service methods.

All of the methods throw com.novell.soa.af.impl.soap.AdminException and java.rmi.RemoteException. To improve readability, the throws clause has been omitted from the method signatures.

Processes

This section provides reference information for each Processes method.

getProcessesByQuery

Used to get information about processes.

Method Signature

```
com.novell.soa.af.impl.soap.ProcessArray
getProcessesByQuery(com.novell.soa.af.impl.soap.T_ProcessInfoQuery query,
int maxRecords)
```

Example

```
        //

   // Query information about processes for a user that are running and
        // have not been approved yet.
        String logic = "AND";
        T_ProcessInfoOrder order = T_ProcessInfoOrder.APPROVAL_STATUS;
        int CHOICE_SIZE = 4;
        Integer approvalStatusInteger = new
Integer(ProcessConstants.PROCESSING);
        Integer processStatusInteger = new
Integer(ProcessConstants.RUNNING);
        //
        // Setup the query with the above params
        T_ProcessInfoQueryChoice [] choice = new
T_ProcessInfoQueryChoice[CHOICE_SIZE];
        choice[0] = new T_ProcessInfoQueryChoice();
        choice[0].setApprovalStatus(approvalStatusInteger);
        choice[1] = new T_ProcessInfoQueryChoice();
        choice[1].setProcessStatus(processStatusInteger);
        choice[2] = new T_ProcessInfoQueryChoice();
        choice[2].setRecipient(recipient);
        choice[3] = new T_ProcessInfoQueryChoice();
        choice[3].setRequestId(requestId);

        int maxRecords = -1;
        T_ProcessInfoQuery processInfoQuery =
                new T_ProcessInfoQuery(T_Logic.fromString(logic), order,
choice);
        ProcessArray processArray =
stub.getProcessesByQuery(processInfoQuery, maxRecords);
```

getProcessesByStatus

Used to get information about processes with a specified status (for example, running processes).

Method Signature

```
public com.novell.soa.af.impl.soap.ProcessArray
getProcessesByStatus(com.novell.soa.af.impl.soap.T_ProcessStatus status)
```

Example

```
      T_ProcessStatus processStatus = T_ProcessStatus.Running;
      //
      // Get processes by status
       ProcessArray processArray =
stub.getProcessesByStatus(processStatus);
       Process [] process = processArray.getProcess();
```

getProcesses

Used to get information about processes, specified by processID.

Method Signature

```
com.novell.soa.af.impl.soap.ProcessArray getProcesses(java.lang.String id,
long time,com.novell.soa.af.impl.soap.T_Operator op, java.lang.String
initiator, java.lang.String recipient)
```

Parameters

Parameter	Description
processId	The process Id (java.lang.String).
creationTime	The time at which the process was started (long).
op	The operator to use. The operators are: EQ - equals LT - less than LE - less than or equal to GT - greater than GE - greater than or equal to
initiator	The initiator of the workflow.
recipient	The recipient of the approval activity.

Example

```
       int processMatchCount = 0;
       T_Operator operator = T_Operator.GT;
       long currentTimeInMillis = System.currentTimeMillis();
       String [] requestIds = requestIdArray.getString();
       //
       // Initialize and start a provisioning request
       HashMap provMap = new HashMap();
       provMap.put(Helper.RECIPIENT, recipient);
       provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
       //
       // Start request
       // Calls method startProvisioningRequest on the provUtils
       // utility object which refers to a utility class that does not
       // ship with the Identity Manager User Application.
String requestId = provUtils.startProvisioningRequest(provMap, null);
       sleep(5);

       Process process = stub.getProcess(requestId);
       if(process != null)
       {
            String processId = process.getProcessId();
            String initiator = process.getInitiator();

            ProcessArray processArray = stub.getProcesses(processId,
currentTimeInMillis, operator, initiator, recipient);
       }
```

getAllProcesses

Used to get information about all running and completed provisioning requests.

Method Signature

```
com.novell.soa.af.impl.soap.ProcessArray getAllProcesses()
```

Example

```
    ProcessArray array = stub.getAllProcesses();
    Process [] processes = array.getProcess();
     if(_process != null)
    {
        sb = new StringBuffer();
        sb.append("\nProcess List:");
        for(int index = 0; index < _process.length; index++)
        {
            String processId = _process[index].getProcessId();
            String approvalStatus = _process[index].getApprovalStatus();
            Calendar completionTime = _process[index].getCompletionTime();
            Calendar creationTime = _process[index].getCreationTime();
            String engineId = _process[index].getEngineId();
            String proxy = _process[index].getProxy();
            String initiator = _process[index].getInitiator();
            String processName = _process[index].getProcessName();
            String processStatus = _process[index].getProcessStatus();
            String p_recipient = _process[index].getRecipient();
            String p_requestId = _process[index].getRequestId();
            int valueOfapprovalStatus =
_process[index].getValueOfApprovalStatus();
            int valueOfprocessStatus =
_process[index].getValueOfProcessStatus();
            String version = _process[index].getVersion();
        }
```

getProcessesArray

Used to limit the number of processes returned. If the limit you specify is less than the system limit, the number you specify is returned. If you exceed the system limit, the Workflow Engine returns the system limit. If the limit you specify is less than or equal to 0, the Workflow Engine returns all processes.

Method Signature

```
com.novell.soa.af.impl.soap.ProcessArray getProcessesArray(int
maxRecords);
```

Example

```
    /**
     * Method to augment the getAllProcesses() method that impose limits
     * on the number of processes returned.
     * @throws TestProgramException
     */
    public void adding_Limits_To_getProcessArray_TestCase()
    throws TestProgramException
    {
        String recipient =
ServiceUtils.getInstance().getLoginData().getUsername(LoginData.RECIPIENT_
TYPE);
        String requestNameToStart =
provUtils.getProvisioningResourceNameForRecipient(recipient,
                "Enable Active Directory");
        //
        // Get the stub
        Provisioning stub =
ServiceUtils.getInstance().getProvisioningStub();
        try
        {
            //
            // Start multiple requests
            final int NUMBER_OF_REQUESTS_TO_START = 2;

            Map map = MapUtils.createAndSetMap(new Object[] {
                    Helper.RECIPIENT, recipient,

IProvisioningConstants.PROVISIONING_REQUEST_TO_START,
requestNameToStart});
            //
            // Start request(s)
            StringArray requestIdArray =
                provUtils.startMultipleProvisioningRequests(map, null,
NUMBER_OF_REQUESTS_TO_START);
            LoggerUtils.sleep(3);
            LoggerUtils.sendToLogAndConsole("Started " +
NUMBER_OF_REQUESTS_TO_START + " provisioning requests");
            //
            // New method to limit the number of processes returned
            //
            // Test Results : maxProcesses <= 0 returns all processes
            //                maxProcesses up to system limit returns
maxProcess count
            //                maxProcesses > system limit returns system
limit
            int maxProcesses = 10;
            ProcessArray processArray =
stub.getProcessesArray(maxProcesses);
            Process [] processes = processArray.getProcess();
            if(processes != null)
            {
                LoggerUtils.sendToLogAndConsole("Process count returned: "
+ processes.length);
```

```
                Assert.assertEquals("Error: Processes returned shouldn't
exceed max count.",
                        maxProcesses, processes.length);
            }
        }
        catch(AdminException error)
        {
            RationalTestScript.logError(error.getReason() );
            throw new TestProgramException(error.getReason() );
        }
        catch(RemoteException error)
        {
            RationalTestScript.logError(error.getMessage() );
            throw new TestProgramException(error.getMessage() );
        }
    }
```

getProcessesById

Used to get information about a specific process, specified by the Process Id.

Method Signature

```
com.novell.soa.af.impl.soap.ProcessArray getProcessesById(java.lang.String
id)
```

Example

```
    Process [] allProcesses = stub.getAllProcesses().getProcess();
    if(allProcesses != null)
    {
          String processId = allProcesses[0].getProcessId;
          ProcessArray array = stub.getProcessesById(processId);
          Process [] processes = array.getProcess();
     }
```

terminate

Used to terminate a running provisioning request.

Method Signature

```
void terminate(java.lang.String requestId,
com.novell.soa.af.impl.soap.T_TerminationType state, java.lang.String
comment)
```

Parameters

Parametere	Description
requestId	The Id of the provisioning request.
state	The reason for terminating the process. The choices are: RETRACT ERROR
comment	Adds a comment about the terminate action.

Example

```
     //
     // Initialize and start a provisioning request
     HashMap provMap = new HashMap();
     provMap.put(Helper.RECIPIENT, recipient);
     provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
     //
     // Start request
     // Calls method startProvisioningRequest on the provUtils
     // utility object which refers to a utility class that does not
     // ship with the Identity Manager User Application.
String requestId = provUtils.startProvisioningRequest(provMap, null);
     sleep(5);
     //
     // Now retract the request
     T_TerminationType terminationType = T_TerminationType.RETRACT;
   stub.terminate(requestId, terminationType, terminationType.getValue() +
" the request");
```

getProcess

Used to get information about a running or completed provisioning request, specified by Request ID.

Method Signature

```
com.novell.soa.af.impl.soap.Process getProcess(java.lang.String requestId)
```

Example

```
     //
     // Initialize and start a provisioning request
     HashMap provMap = new HashMap();
     provMap.put(Helper.RECIPIENT, recipient);
     provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
     //
     // Start request
```

```
       // Calls method startProvisioningRequest on the provUtils
       // utility object which refers to a utility class that does not
       // ship with the Identity Manager User Application.
     String requestId = provUtils.startProvisioningRequest(provMap, null);
     sleep(5);

    Process process = stub.getProcess(requestId);
    if(process != null)
    {
        boolean bMatchProcess = false;
        if( (recipient.compareTo(process.getRecipient()) == 0) &&
(requestId.compareTo(process.getRequestId()) == 0) )
        {
            bMatchProcess = true;
        }
        if(bMatchProcess)
        {
            String msg = "Found process with requestId : " + requestId;
            LoggerUtils.sendToLogAndConsole(msg);
        }
        //
        // Assert if we could not find a match
        Assert.assertTrue("Could not find process with request id: " +
requestId, bMatchProcess);
    }
```

getProcessesByCreationTime

Used to get information about processes created between the current time and the time at which the workflow process was created.

Method Signature

```
com.novell.soa.af.impl.soap.ProcessArray getProcessesByCreationTime(long
time, com.novell.soa.af.impl.soap.T_Operator op)
```

Parameters

Parameter	Description
creationTime	The time at which the process was started.
op	The operator to use. The operators are: EQ - equals LT - less than LE - less than or equal to GT - greater than GE - greater than or equal to

Example

```
     T_Operator operator = T_Operator.GT;
    //
    // Get processes with operator relative to the current time
    long currentTime = System.currentTimeMillis();//
currentDateTime.getTime();
    ProcessArray processArray =
stub.getProcessesByCreationTime(currentTime, operator);
```

getProcessesByApprovalStatus

Used to get information about processes with a specified approval status (Approved, Denied, or Retracted).

Method Signature

```
com.novell.soa.af.impl.soap.ProcessArray
getProcessesByApprovalStatus(com.novell.soa.af.impl.soap.T_ApprovalStatus
status)
```

Example

```
    T_ApprovalStatus approvalStatus = T_ApprovalStatus.Approved;
    //
    // Get all the processes based upon approval status above
    ProcessArray processArray =
stub.getProcessesByApprovalStatus(approvalStatus);
    Process [] processes = processArray.getProcess();
```

getProcessesByRecipient

Used to get information about processes that have a specific recipient Id.

Method Signature

```
com.novell.soa.af.impl.soap.ProcessArray
getProcessesByRecipient(java.lang.String recipient)
```

Example

```
    String recipient = "cn=ablake,ou=users,ou=idmsample-komodo,o=netiq";

    //
    // Get processes by recipient
    ProcessArray processArray = stub.getProcessesByRecipient(recipient);
    Process [] process = processArray.getProcess();
```

getProcessesByInitiator

Used to get information about processes that have a specific initiator Id.

Method Signature

```
com.novell.soa.af.impl.soap.ProcessArray
getProcessesByInitiator(java.lang.String initiator)
```

Example

```
    String initiator = "cn=admin,ou=idmsample-komodo,o=netiq";

    //
    // Get processes by initiator
    ProcessArray processArray = stub.getProcessesByInitiator(initiator);
    Process [] process = processArray.getProcess();
```

setResult

Used to set the entitlement result (approval status) of a previously completed provisioning request.

Method Signature

```
void setResult(java.lang.String requestId,
com.novell.soa.af.impl.soap.T_EntitlementState state,
com.novell.soa.af.impl.soap.T_EntitlementStatus status, java.lang.String
message)
```

Parameters

Parameter	Description
requestId	The Id of the provisioning request.
state	The state of the provisioning request. The possible values are: Unknown Granted Revoked
status	The status of the provisioning request. The possible values are: Unknown Success Warning Error Fatal Submitted
message	A message about the entitlement result.

Example

```
     //
     // Initialize and start a provisioning request
     HashMap provMap = new HashMap();
     provMap.put(Helper.RECIPIENT, recipient);
     provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
     //
     // Start request
     // Calls method startProvisioningRequest on the provUtils
     // utility object which refers to a utility class that does not
     // ship with the Identity Manager User Application.
     String requestId = provUtils.startProvisioningRequest(provMap, null);
     sleep(5);

     //
     // Get the process id for this running process
     Process process = stub.getProcess(requestId);
     String processId = null;
     if (process != null)
        processId = process.getProcessId();
    //
    // Reset the state of the provisioning request
    T_EntitlementState newEntitlementState =
T_EntitlementState.Revoked;
    T_EntitlementStatus newEntitlementStatus = T_EntitlementStatus.Success;
    String comment = "Revoked the provisioning request";
    stub.setResult(processId, newEntitlementState, newEntitlementStatus,
comment);
```

getProcessesByCreationInterval

Used to get information about processes started between two specified times.

Method Signature

```
com.novell.soa.af.impl.soap.ProcessArray
getProcessesByCreationInterval(long start, long end)
```

Parameters

Parameter	Description
startTime	The start time (YYYY/MM/DD).
endTime	The end time (YYYY/MM/DD).

Example

```
     long startTime = System.currentTimeMillis();
     //
     // Initialize and start a provisioning request
     HashMap provMap = new HashMap();
     provMap.put(Helper.RECIPIENT, recipient);
     provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
     //
     // Start request
     // Calls method startProvisioningRequest on the provUtils
     // utility object which refers to a utility class that does not
     // ship with the Identity Manager User Application.
     String requestId = provUtils.startProvisioningRequest(provMap, null);
     sleep(5);

    long endTime = System.currentTimeMillis();
    //
    // Get all the processes between the start and end time    ProcessArray
processArray = stub.getProcessesByCreationInterval(startTime, endTime);
    Process [] processes = processArray.getProcess();
```

Provisioning

This section provides reference information for each Provisioning method.

multiStart

Used to start a workflow request for each specified recipient.

Method Signature

```
com.novell.soa.af.impl.soap.StringArray multiStart(java.lang.String
processId, com.novell.soa.af.impl.soap.StringArray recipients,
com.novell.soa.af.impl.soap.DataItemArray items)
```

Parameters

Parameter	Description
processId	The Id of the provisioning request to start.
recipients	The DN of each recipient.
dataItem	The list of data items for the provisioning request.

Example

```
     ProvisioningRequestArray requestArray =
stub.getAllProvisioningRequests(recipient);

    //
    // If there are some then,
    if(requestArray != null)
    {
        String Id = " ";
        StringArray requestIdStringArray = null;
        String [] listOfRecipients = {recipient, addressee};
        //
        // Select a provisioning resource
        String requestNameToStart = "Enable Active Directory Account (Mgr
Approve-No Timeout)";
        //
        // Loop thru and find the request that we want to start
        ProvisioningRequest [] requests =
requestArray.getProvisioningrequest();
        for(int index = 0; index < requests.length; index++)
        {
            //
            // Is this the name of the request to start?
            if(requests[index].getName().compareTo(requestNameToStart) ==
0)
            {
                //
                // Get the current associated data items. Replicate a new
                // dataitem array excluding the null values.
                Id = requests[index].getId();
                DataItem [] dataItem =
requests[index].getItems().getDataitem();
                if(dataItem != null)
                // Call method replicateDataItemArray on the
                // provUtils utility object, which refers to a
                // utility class that does not ship with the
                // Identity Manager User Application.
                {
                    DataItemArray newDataItemArray =
provUtils.replicateDataItemArray(dataItem);
                    //
                    // Create a string array initializing with multiple
recipients
                    StringArray listOfRecipientsStringArray = new
StringArray(listOfRecipients);
                    //
                   // Start the request for multiple recipients
                    logStep("Calling stub.multiStart(" + Id +
",listOfRecipientsStringArray,newDataItemArray)");
                    requestIdStringArray = stub.multiStart(Id,
listOfRecipientsStringArray, newDataItemArray);
            }
        }
}
```

start

Used to start a provisioning request.

Method Signature

```
java.lang.String start(java.lang.String processId, java.lang.String
recipient, com.novell.soa.af.impl.soap.DataItemArray items)
```

Parameters

Parameter	Description
processId	The Id of the provisioning request to start.
recipient	The DN of each recipient.
dataItem	The list of data items for the provisioning request.

Example

```
       //
       // Initialize and start a provisioning request
       HashMap provMap = new HashMap();
       provMap.put(Helper.RECIPIENT, recipient);
       provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
       //
       // Start request
       // Calls method startProvisioningRequest on the provUtils
       // utility object which refers to a utility class that does not
       // ship with the Identity Manager User Application.
       String requestId = provUtils.startProvisioningRequest(provMap,
null);
       sleep(5);
```

The example above calls the startProvisioningRequest method. This method is not part of the IDM User Application. We show it here to finish illustrating the example:

```
          /**
           *Method to start a provisioning request using the supplied
           *Map and dataitem object. Handling of digital certificate
           *resources is also handled.
           * @param _map
           * @param _in_dataItem
           * @return String
           * @throws TestProgrammException
           */
          public String startProvisioningRequest(Map _map, DataItem []
          _in_dataItem) throws TestProgramException
         {
            String requestId = null;
            try
            {
               String recipient  =(String)_map.get(Helper.RECIPIENT);
```

```
                String requestToStart =
(String)_map.get(IProvisioningConstants.PROVISIONING_REQUEST_TO_START);
                String proxyUser
=(String)_map.get(IWorkFlowConstants.PROXY_USER);
                String digitalSignature =
String)_map.get(IDigitalSignatureConstants.DIGITAL_SIGNATURE);
                RationalTestScript.logInfo("Step: Calling
startProvisioningRequest(_map)");
                //
                //Get the stub
                Provisioning stub =
ServiceUtils.getInstance().getProvisioningStub();
                //
        //Get all the available resource requests for the recipient
        RationalTestScript.logInfo("Step: Calling
stub.getAllProvisioningRequests(" + recipient + ")");
        ProvisioningRequestArray requestArray =
stub.getAllProvisioningRequests(recipient);

        if(requestArray != null)
        {
            //
            //Get the provisioning request from the array
            ProvisioningRequest request =
getProvisioningRequestFromArray(requestArray, requestToStart);
            if(request != null)
        {
        DataItem [] dataItem = null;
        DataItemArray newDataItemArray = null;
        //
        // If the supplied data item is null then just replicate
        // what currently exists with the request.
        if(_in_dataItem == null)
        {
            //
            // Use the current data item associated with the request
            dataItem = request.getItems().getDataitem();
            if(dataItem != null)
            {
                newDataItemArray = replicateDataItemArray(dataItem);
            }
        }
        else
        {
            //
            // Set the incoming data item array
            newDataItemArray = new DataItemArray();
            newDataItemArray.setDataitem(_in_dataItem);
        }
```

```
       //
       // Start the Provisioning request for the recipient
       if(proxyUser == null && digitalSignature == null)
          {
              RationalTestScript.logInfo("Step: Calling stub.start(" +
request.getId() + "," + recipient + "dataItemArray)");
                        requestId = stub.start(
                                request.getId(),
                                recipient,
                                newDataItemArray);
          }
           else if(proxyUser != null && digitalSignature == null)
           }
                 .

           .
           .
```

getAllProvisioningRequests

Used to return an array of available provisioning requests.

Method Signature

```
com.novell.soa.af.impl.soap.ProvisioningRequestArray
getAllProvisioningRequests(java.lang.String recipient)
```

Example

```
    //

    // Get all the provisioning requests for this recipient

    ProvisioningRequestArray provReqArray =
stub.getAllProvisioningRequests(recipient);
    ProvisioningRequest [] provRequest =
provReqArray.getProvisioningrequest();
    if(provRequest != null)
    {
        String description = provRequest[0].getDescription();
        String category = provRequest[0].getCategory();
        String digitialSignatureType =
provRequest[0].getDigitalSignatureType();
        String requestId = provRequest[0].getId();
        DataItemArray itemArray = provRequest[0].getItems();
        String legalDisclaimer = provRequest[0].getLegalDisclaimer();
        String name = provRequest[0].getName();
        String operation = provRequest[0].getOperation();
    }
```

getProvisioningRequests

Used to return an array of provisioning requests for a specified category and operation.

Method Signature

```
com.novell.soa.af.impl.soap.ProvisioningRequestArray
getProvisioningRequests(java.lang.String recipient, java.lang.String
category, java.lang.String operation)
```

Parameters

Parameter	Description
recipient	The recipient of the provisioning request.
category	The category of the provisioning request.
operation	The provisioning request operation (0=Grant,1=Revoke, 2=Both)

Example

```
    String operation = IProvisioningRequest.GRANT;
    try
    {
        //
        // Get the stub
        Provisioning stub =
ServiceUtils.getInstance().getProvisioningStub();
        logStep("Calling stub.getProvisioningCategories()");
        StringArray categoriesStringArray =
stub.getProvisioningCategories();
        String [] categories = categoriesStringArray.getString();
        //
        // Loop thru and get the provisioning requests for each category
        for(int index = 0; index < categories.length; index++)
        {
            //
            // Get the provisioning request based upon recipient
            logStep("Calling stub.getProvisioningRequests(" + recipient +
"," + categories[index] + "," + operation + ")");
            ProvisioningRequestArray provRequestArray =
stub.getProvisioningRequests(recipient, categories[index], operation);
            ProvisioningRequest [] provRequests =
provRequestArray.getProvisioningrequest();
        }
```

getProvisioningCategories

Used to get the list of available provisioning categories.

Method Signature

```
com.novell.soa.af.impl.soap.StringArray getProvisioningCategories()
```

Example

```
       StringArray categoriesStringArray =
stub.getProvisioningCategories();
       String [] categories = categoriesStringArray.getString();
```

startAsProxy

Used to start a workflow as a proxy.

Method Signature

```
java.lang.String startAsProxy(java.lang.String processId, java.lang.String
recipient, com.novell.soa.af.impl.soap.DataItemArray items,
java.lang.String proxyUser)
```

Parameters

Parameter	Description
processId	The Id of the provisioning request.
recipient	The recipient of the provisioning request.
Items	The data items for the provisioning request.
proxyUser	The DN of the proxy user.

Example

```
    ProvisioningRequestArray requestArray =
stub.getAllProvisioningRequests(recipient);
    //
    // If there are some then,
    if(requestArray != null)
    {
        String Id = " ";
        String requestId = " ";
        String requestNameToStart = "Enable Active Directory Account (Mgr
Approve-No Timeout)";
        //
        // Loop thru and find the request that we want to start
        ProvisioningRequest [] requests =
requestArray.getProvisioningrequest();
        for(int index = 0; index < requests.length; index++)
        {
            //
            // Is this the name of the request to start?
            if(requests[index].getName().compareTo(requestNameToStart) ==
0)
            {
                //
                // Get the current associated data items. Replicate a new
                // dataitem array excluding the null values.
```

```
                Id = requests[index].getId();
                DataItem [] dataItem =
requests[index].getItems().getDataitem();
                if(dataItem != null)
                {
                    // Call method replicateDataItemArray on the
                    // provUtils utility object, which refers to a
                    // utility class that does not ship with the
                    // Identity Manager User Application.
                    DataItemArray newDataItemArray =
provUtils.replicateDataItemArray(dataItem);
                    //
                    // Start the Provisioning request for the recipient
                    logStep("Calling stub.startAsProxy(" + Id + "," +
recipient + ",newDataItemArray," + proxyUser + ")");
                    requestId = stub.startAsProxy(Id, recipient,
newDataItemArray, proxyUser);
                }
            }
        }
    }
```

getProvisioningStatuses

Used to get the status of provisioning requests.

Method Signature

```
com.novell.soa.af.impl.soap.ProvisioningStatusArray
getProvisioningStatuses(com.novell.soa.af.impl.soap.T_ProvisioningStatusQu
ery query, int maxRecords)
```

Parameters

Parameter	Description
query	Used to specify the provisioning status query. The query has the following components: ◆ `choice` - the parameters used to filter the results. You can specify multiple parameters. The possible parameters are: Recipient - a DN RequestID ActivityID Status (an integer) State (an integer) ProvisioningTime (YYYY/MM/DD) ResultTime (YYYY/MM/DD) ◆ `logic` - AND or OR ◆ `order` - the order in which to sort the results. Possible values for `order` are: ACTIVITY_ID RECIPIENT PROVISIONING_TIME RESULT_TIME STATE STATUS REQUEST_ID MESSAGE
maxRecords	Used to specify maximum number of records to retrieve. A value of -1 returns unlimited records.

Example

```
     //
     // Initialize and start a provisioning request
     HashMap provMap = new HashMap();
     provMap.put(Helper.RECIPIENT, recipient);
     provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
     //
     // Start request
     // Calls method startProvisioningRequest on the provUtils
     // utility object which refers to a utility class that does not
     // ship with the Identity Manager User Application.
     String requestId = provUtils.startProvisioningRequest(provMap, null);
     sleep(5);
     //
     //
    T_ProvisioningStatusQueryChoice [] choice = new
```

```
T_ProvisioningStatusQueryChoice[3];
    choice[0] = new T_ProvisioningStatusQueryChoice();
    choice[0].setRecipient(recipient);
    choice[1] = new T_ProvisioningStatusQueryChoice();
    choice[1].setRequestId(requestId);
    choice[2] = new T_ProvisioningStatusQueryChoice();
    choice[2].setStatus(new Integer(ProcessConstants.PROCESSING) );
    //
    // Initialize the query
    T_ProvisioningStatusQuery query = new
T_ProvisioningStatusQuery(T_Logic.AND, T_ProvisioningStatusOrder.STATUS,
choice);
    //
    // Make the query
    StringBuffer sb = new StringBuffer();
    int maxRecords = -1;

    ProvisioningStatusArray provStatusArray =
stub.getProvisioningStatuses(query, maxRecords);
```

startWithDigitalSignature

Used to start a workflow and specify that a digital signature is required.

Method Signature

```
java.lang.String startWithDigitalSignature(java.lang.String processId,
java.lang.String recipient, com.novell.soa.af.impl.soap.DataItemArray
items, java.lang.String digitalSignature,
com.novell.soa.af.impl.soap.SignaturePropertyArray
digitalSignaturePropertyArray)
```

Parameters

Parameter	Description
processId	The request identifier.
recipient	The request recipient.
items	The data items for the provisioning request.
digital signature	The digital signature.
digitalSignaturePropertyArray.	The digital signature property map.

Example

```
    String recipient =
ServiceUtils.getInstance().getLoginData().getUsername(LoginData.RECIPIENT_
TYPE);
    //
    // Get the digital signature string for admin
    String digitalSignature =
DigitalSignatureUtils.getDigitalSignatureFromFile(IDigitalSignatureConstan
ts.ADMIN_DIGITAL_SIGNATURE_FILENAME);


    ProvisioningRequestArray requestArray =
stub.getAllProvisioningRequests(recipient);
    //
    // If there are some then,

   if(requestArray != null)
    {
        String Id = " ";
        String requestId = " ";
        String requestNameToStart = "Enable Active Directory Account (Mgr
Approve-No Timeout)";
        //
        // Loop thru and find the request that we want to start
        ProvisioningRequest [] requests =
requestArray.getProvisioningrequest();
        for(int index = 0; index < requests.length; index++)
        {
            //
            // Is this the name of the request to start?
            if(requests[index].getName().compareTo(requestNameToStart) ==
0)
            {
                //
                // Get the current associated data items. Replicate a new
                // dataitem array excluding the null values.
                Id = requests[index].getId();
                DataItem [] dataItem =
requests[index].getItems().getDataitem();
                if(dataItem != null)
                {
                    // Call method replicateDataItemArray on the
                    // provUtils utility object, which refers to a
                    // utility class that does not ship with the
```

```
                    // Identity Manager User Application.
                    DataItemArray newDataItemArray =
provUtils.replicateDataItemArray(dataItem);
                    //
                    // Start a digitally signed provisioning resource for
the recipient
                    requestId =
stub.startWithDigitalSignature(request.getId(), recipient,
newDataItemArray, digitalSignature, null); // Don't get any property values
(optional)
                }
            }
        }
    }
```

startAsProxyWithDigitalSignature

Used to start a workflow using a proxy for the initiator, and specify that a digital signature is required.

Method Signature

```
java.lang.String startAsProxyWithDigitalSignature(java.lang.String
processId, java.lang.String recipient,
com.novell.soa.af.impl.soap.DataItemArray items, java.lang.String
digitalSignature, com.novell.soa.af.impl.soap.SignaturePropertyArray
digitalSignaturePropertyArray, java.lang.String proxyUser)
```

Parameters

Parameter	Description
processId	The request identifier.
recipient	The request recipient.
items	The data items for the provisioning request.
digital signature	The digital signature.
digitalSignaturePropertyArray.	The digital signature property map.
proxyUser	The DN of the proxy user.

Example

```
    //
    // Get the digital signature string for admin
    String digitalSignature =
DigitalSignatureUtils.getDigitalSignatureFromFile(IDigitalSignatureConstan
ts.ADMIN_DIGITAL_SIGNATURE_FILENAME);


    ProvisioningRequestArray requestArray =
stub.getAllProvisioningRequests(recipient);
    //
    // If there are some then,
    if(requestArray != null)
    {
        String Id = " ";
        String requestId = " ";
        String requestNameToStart = "Enable Active Directory Account (Mgr
Approve-No Timeout)";
        //
        // Loop thru and find the request that we want to start
        ProvisioningRequest [] requests =
requestArray.getProvisioningrequest();
        for(int index = 0; index < requests.length; index++)
        {
            //
            // Is this the name of the request to start?
            if(requests[index].getName().compareTo(requestNameToStart) ==
0)
            {
                //
                // Get the current associated data items. Replicate a new
                // dataitem array excluding the null values.
                Id = requests[index].getId();
                DataItem [] dataItem =
requests[index].getItems().getDataitem();
                if(dataItem != null)
                {
                    // Call method replicateDataItemArray on the
                    // provUtils utility object, which refers to a
                    // utility class that does not ship with the
                    // Identity Manager User Application.
                    DataItemArray newDataItemArray =
provUtils.replicateDataItemArray(dataItem);
                    //
                    // Start a digitally signed provisioning resource as
proxy for the recipient

                     requestId =
stub.startAsProxyWithDigitalSignature(request.getId(), recipient,
newDataItemArray, digitalSignature, null, proxyUser);
                }
            }
        }
    }
```

startWithCorrelationId

Used to start a workflow with a correlation ID. The correlation ID provides a way to track a set of related workflow processes. When started with this method, workflow processes can be queried and sorted by correlation ID.

Method Signature

```
java.lang.String startWithCorrelationId(java.lang.String processId,
java.lang.String recipient, com.novell.soa.af.impl.soap.DataItemArray
items, java.lang.String signature,
com.novell.soa.af.impl.soap.SignaturePropertyArray props, java.lang.String
proxyUser, java.lang.String correlationId)
     throws com.novell.soa.af.impl.soap.AdminException,
java.rmi.RemoteException;
```

Parameters

Parameter	Description
processId	The request identifier.
recipient	The request recipient.
items	The data items for the provisioning request.
digital signature	The digital signature.
digitalSignaturePropertyArray	The digital signature property map.
proxyUser	The DN of the proxy user.
correlationID	The string that identities the correlation ID. The correlation ID cannot be longer than 32 characters.

Work Entries

This section provides reference information for each Work Entries method.

forward

Used to forward a task to the next activity in the workflow with the appropriate action (approve, deny, refuse).

Method Signature

```
void forward(java.lang.String wid, com.novell.soa.af.impl.soap.T_Action
action, com.novell.soa.af.impl.soap.DataItemArray items, java.lang.String
comment)
```

Parameters

Parameter	Description
wid	The work Id.
action	The action to take (approve, deny, refuse).
items	The data items required by the workflow.
comment	The comment.

Example

```
     //
     // Initialize and start a provisioning request
     HashMap provMap = new HashMap();
     provMap.put(Helper.RECIPIENT, recipient);
     provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
     //
     // Start request
     // Calls method startProvisioningRequest on the provUtils
     // utility object which refers to a utility class that does not
     // ship with the Identity Manager User Application.
     String requestId = provUtils.startProvisioningRequest(provMap, null);
     sleep(5);
     //
    // Get the process id for this running process
    Process process = stub.getProcess(requestId);
    String processId = null;
    if(process != null)
        processId = process.getProcessId();

    T_Action action = T_Action.APPROVE;

    T_Logic logic = T_Logic.AND;

    T_WorkEntryOrder workEntryOrder = T_WorkEntryOrder.REQUEST_ID;

    T_WorkEntryQueryChoice [] workEntryqueryChoice =  new
T_WorkEntryQueryChoice[3];
    workEntryqueryChoice[0] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[0].setRecipient(recipient);
    workEntryqueryChoice[1] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[1].setRequestId(requestId);
    workEntryqueryChoice[2] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[2].setProcessId(processId);
    //
    // Create work entry query
    T_WorkEntryQuery query = new T_WorkEntryQuery(logic, _workEntryOrder,
workEntryqueryChoice);
    //
    // Get all work entries (max records)
    WorkEntryArray workEntryArray = stub.getWorkEntries(query, -1);
```

```
    WorkEntry [] workEntry = workEntryArray.getWorkentry();

    if(workEntry != null

    {
         for(int wIndex = 0; wIndex < workEntry.length; wIndex++)
        {
            String workId = workEntry[wIndex].getId();
            //
            //
            LoggerUtils.sendToLogAndConsole("Forwarding : " +
workEntry[wIndex].getActivityName() + " work id: " + workId);
    //
           // Get the dataitem for this item of work
            DataItemArray dataItemArray = stub.getWork(workId);
            DataItem [] dataItem = dataItemArray.getDataitem();
            DataItemArray newDataItemArray = null;
            if(dataItem != null)
                // Call method replicateDataItemArray on the
                // provUtils utility object, which refers to a
                // utility class that does not ship with the
                // Identity Manager User Application.
                newDataItemArray =
provUtils.replicateDataItemArray(dataItem);
            else
                throw new TestProgramException("DataItem is null.");
            //
            // Claim request for recipient
            String comment = _action.toString() + " this request: " +
requestId + " for " + recipient;
            stub.forward(workId, _action, newDataItemArray, comment);
        }

    }
```

reassignWorkTask

Used to reassign a task from one user to another.

Method Signature

```
void reassignWorkTask(java.lang.String wid, java.lang.String addressee,
java.lang.String comment)
```

Parameters

Parameter	Description
wid	The Id of the task.
addressee	The addressee of the task.
comment	A comment about the task.

Example

```
       //
       // Initialize and start a provisioning request
       HashMap provMap = new HashMap();
       provMap.put(Helper.RECIPIENT, recipient);
       provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
       //
       // Start request
       // Calls method startProvisioningRequest on the provUtils
       // utility object which refers to a utility class that does not
       // ship with the Identity Manager User Application.
       String requestId = provUtils.startProvisioningRequest(provMap,
null);
       sleep(5);
      //
      // Get the process id for this running process
      Process process = stub.getProcess(requestId);
      if(process != null)
      {
            String processId = process.getProcessId();
            String initiator = process.getInitiator();
            //
            // Setup for the query
            HashMap map = new HashMap();
            map.put(Helper.REQUESTID, requestId);
            map.put(Helper.RECIPIENT, recipient);
            map.put(Helper.PROCESSID, processId);
            map.put(Helper.INITIATOR, initiator);
            WorkEntry [] workEntry =
workEntryUtils.getWorkEntriesUsingQuery(map, T_WorkEntryOrder.REQUEST_ID,
T_Logic.AND);

            if(workEntry == null)
                throw new TestProgramException("Work list is empty.");
            //
            // Reassign the work entry from recipient to the addressee
            //
            // Should only be one item
            String reassignComment = null;
            String workId = workEntry[0].getId();
            if(workId != null)
            {
                //
                // Reassign work entry(s) to addressee
                reassignComment = "Reassigning work entry " + workId + "
from " + recipient + " to " + addressee;
                stub.reassign(workId, addressee, reassignComment);
                LoggerUtils.sendToLogAndConsole("Reassign work entry " +
workId + " from " + recipient + " to " + addressee);
            }
    }
```

getWork

Used to retrieve data items for a work entry identified by the Id (UUID) of a task.

Method Signature

```
com.novell.soa.af.impl.soap.DataItemArray getWork(java.lang.String workId)
```

Example

```
       //
       // Initialize and start a provisioning request
       HashMap provMap = new HashMap();
       provMap.put(Helper.RECIPIENT, recipient);
       provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
       //
       // Start request
       // Calls method startProvisioningRequest on the provUtils
       // utility object which refers to a utility class that does not
       // ship with the Identity Manager User Application.
       String requestId = provUtils.startProvisioningRequest(provMap,
null);
       sleep(5);
       //
       // Get the process id for this running process
      Process process = stub.getProcess(requestId);
      if(process != null)
      {
            String processId = process.getProcessId();
            String initiator = process.getInitiator();
            //
            // Setup for the query
            HashMap map = new HashMap();
            map.put(Helper.REQUESTID, requestId);
            map.put(Helper.RECIPIENT, recipient);
            map.put(Helper.PROCESSID, processId);
            map.put(Helper.INITIATOR, initiator);
            WorkEntry [] workEntry =
workEntryUtils.getWorkEntriesUsingQuery(map, T_WorkEntryOrder.REQUEST_ID,
T_Logic.AND);
            //
            // Do assertion here
            Assert.assertNotNull("WorkEntry is null for recipient : " +
recipient + " with request id : " + requestId, workEntry);
          DataItemArray dataItemArray = stub.getWork(workEntry[0].getId()
);
            DataItem [] dataItem = dataItemArray.getDataitem();
            if(dataItem != null)
                LoggerUtils.sendToLogAndConsole(dataItem[0].getName());
      }
```

forwardWithDigitalSignature

Used to forward a provisioning request with a digital signature and optional digital signature properties. For example, this can be used by an administrator to force a user-facing activity to be approved, denied or refused.

Method Signature

```
void forwardWithDigitalSignature(java.lang.String wid,
com.novell.soa.af.impl.soap.T_Action action,
com.novell.soa.af.impl.soap.DataItemArray items, java.lang.String comment,
java.lang.String digitalSignature,
com.novell.soa.af.impl.soap.SignaturePropertyArray
digitalSignaturePropertyArray)
```

Parameters

Parameter	Description
wid	The workId.
action	The action to take (approve, deny, refuse).
items	The data items required by the workflow.
comment	A comment about the action.
digitalSignature	The digital signature.
digitalSignaturePropertyArray	The digital signature property map.

Example

```
     //
     // Initialize and start a provisioning request
     HashMap provMap = new HashMap();
     provMap.put(Helper.RECIPIENT, recipient);
     provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
     //
     // Start request
     // Calls method startProvisioningRequest on the provUtils
     // utility object which refers to a utility class that does not
     // ship with the Identity Manager User Application.
     String requestId = provUtils.startProvisioningRequest(provMap, null);
     sleep(5);
     //
    // Get the process id for this running process
    Process process = stub.getProcess(requestId);
    String processId = null;
    if(process != null)
        processId = process.getProcessId();

    T_Action action = T_Action.APPROVE;
```

```
    T_Logic logic = T_Logic.AND;

    T_WorkEntryOrder workEntryOrder = T_WorkEntryOrder.REQUEST_ID;


 // Get the digital signature string for admin
 String digitalSignature =
DigitalSignatureUtils.getDigitalSignatureFromFile(IDigitalSignatureConstan
ts.ADMIN_DIGITAL_SIGNATURE_FILENAME);

    T_WorkEntryQueryChoice [] workEntryqueryChoice =  new
T_WorkEntryQueryChoice[3];
    workEntryqueryChoice[0] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[0].setRecipient(recipient);
    workEntryqueryChoice[1] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[1].setRequestId(requestId);
    workEntryqueryChoice[2] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[2].setProcessId(processId);
    //
    // Create work entry query
    T_WorkEntryQuery query = new T_WorkEntryQuery(logic, _workEntryOrder,
workEntryqueryChoice);
    //
    // Get all work entries (max records)
    WorkEntryArray workEntryArray = stub.getWorkEntries(query, -1);

    WorkEntry [] workEntry = workEntryArray.getWorkentry();

    if(workEntry != null

    {
         for(int wIndex = 0; wIndex < workEntry.length; wIndex++)
        {
            String workId = workEntry[wIndex].getId();
            //
            //
            LoggerUtils.sendToLogAndConsole("Forwarding : " +
workEntry[wIndex].getActivityName() + " work id: " + workId);
            //
            // Get the dataitem for this item of work
            DataItemArray dataItemArray = stub.getWork(workId);
            DataItem [] dataItem = dataItemArray.getDataitem();
            DataItemArray newDataItemArray = null;
            if(dataItem != null)
                  // Call method replicateDataItemArray on the
                  // provUtils utility object, which refers to a
                  // utility class that does not ship with the
```

```
                // Identity Manager User Application.
               newDataItemArray =
provUtils.replicateDataItemArray(dataItem);
            else
                throw new TestProgramException("DataItem is null.");
            //
            // Claim request for recipient
            String comment = _action.toString() + " this request: " +
requestId + " for " + recipient;
            stub.forwardWithDigitalSignature(workId, _action,
newDataItemArray, comment, digitalSignature, null);
  }

    }
```

forwardAsProxy

Used to forward a provisioning request. For example, this can be used by an administrator to force a user-facing activity to be approved, denied or refused.

Method Signature

```
void forwardAsProxy(java.lang.String wid,
com.novell.soa.af.impl.soap.T_Action action,
com.novell.soa.af.impl.soap.DataItemArray items, java.lang.String comment,
java.lang.String proxyUser)
```

Parameters

Parameter	Description
wid	The workId (activity Id).
action	The action to take (approve, deny, refuse).
items	The data items required by the workflow.
comment	The comment to add to the activity.
proxyUser	The DN of the proxy user.

Example

```
     //
     // Initialize and start a provisioning request
     HashMap provMap = new HashMap();
     provMap.put(Helper.RECIPIENT, recipient);
     provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
     //
     // Start request
     // Calls method startProvisioningRequest on the provUtils
     // utility object which refers to a utility class that does not
     // ship with the Identity Manager User Application.
     String requestId = provUtils.startProvisioningRequest(provMap, null);
     sleep(5);
     //
    // Get the process id for this running process
    Process process = stub.getProcess(requestId);
    String processId = null;
    if(process != null)
        processId = process.getProcessId();

    T_Action action = T_Action.APPROVE;

    T_Logic logic = T_Logic.AND;

    T_WorkEntryOrder workEntryOrder = T_WorkEntryOrder.REQUEST_ID;


    T_WorkEntryQueryChoice [] workEntryqueryChoice =  new
T_WorkEntryQueryChoice[3];
    workEntryqueryChoice[0] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[0].setRecipient(recipient);
    workEntryqueryChoice[1] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[1].setRequestId(requestId);
    workEntryqueryChoice[2] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[2].setProcessId(processId);
    //
    //  work entry query
    T_WorkEntryQuery query = new T_WorkEntryQuery(logic, _workEntryOrder,
workEntryqueryChoice);
    //
    // Get all work entries (max records)
    WorkEntryArray workEntryArray = stub.getWorkEntries(query, -1);

    WorkEntry [] workEntry = workEntryArray.getWorkentry();

    if(workEntry != null

    {
         for(int wIndex = 0; wIndex < workEntry.length; wIndex++)
        {
            String workId = workEntry[wIndex].getId();
            //
            //
```

```
            LoggerUtils.sendToLogAndConsole("Forwarding : " +
workEntry[wIndex].getActivityName() + " work id: " + workId);
            //
            // Get the dataitem for this item of work
            DataItemArray dataItemArray = stub.getWork(workId);
            DataItem [] dataItem = dataItemArray.getDataitem();
            DataItemArray newDataItemArray = null;
            if(dataItem != null)
                // Call method replicateDataItemArray on the
                // provUtils utility object, which refers to a
                // utility class that does not ship with the
                // Identity Manager User Application.
                newDataItemArray =
provUtils.replicateDataItemArray(dataItem);
            else
                throw new TestProgramException("DataItem is null.");
            //
            // Claim request for recipient
            String comment = _action.toString() + " this request: " +
requestId + " for " + recipient;
            String proxyUser =
ServiceUtils.getInstance().getLoginData().getUsername(LoginData.PROXY_TYPE
);
          stub.forwardAsProxy(workId, _action, newDataItemArray, comment,
proxyUser);        }

    }
```

unclaim

Used to unclaim a provisioning request. This method only works if the request was claimed in the User Application. You cannot unclaim a request once it has been forwarded using the SOAP interface, because the `forward` API method (see “forward” on page 295) claims and forwards in one operation.

Method Signature

```
void unclaim(java.lang.String wid, java.lang.String comment)
```

Parameters

Parameter	Description
workId	The Id of the activity to unclaim.
comment	A comment about the action.

Example

```
    // Action and Approval Types
    final int SELECTED_ACTION = 0; final int CLAIMED_SELECTED_ACTION = 0;
    T_Action [] action = {T_Action.APPROVE, T_Action.REFUSE,
T_Action.DENY};
    T_ApprovalStatus [] claimedAction = {T_ApprovalStatus.Approved,
T_ApprovalStatus.Retracted, T_ApprovalStatus.Denied};
    //
    // Get the process id for this running process
    Process process = stub.getProcess(requestId);
    String processId = null;
    if(process != null)
        processId = process.getProcessId();

    HashMap map = new HashMap();
    map.put(Helper.REQUESTID, requestId);
    map.put(Helper.RECIPIENT, recipient);
    map.put(Helper.PROCESSID, processId);
    //
    // Claim the request
    WorkEntry workEntry = workEntryUtils.claimWorkEntry(map,
action[SELECTED_ACTION]);
    if(workEntry != null)
    {
        //
        // Now unclaim the entry
        String workId = workEntry.getId();
       stub.unclaim(workId, "Unclaiming this work item : " + workId + " for
request id : " + requestId);
   }
```

forwardAsProxyWithDigitalSignature

Used to forward a provisioning request with a digital signature and digital signature properties. For example, this can be used by an administrator to force a user-facing activity to be approved, denied or refused.

Method Signature

```
void forwardAsProxyWithDigitalSignature(java.lang.String wid,
com.novell.soa.af.impl.soap.T_Action action,
com.novell.soa.af.impl.soap.DataItemArray items, java.lang.String comment,
java.lang.String digitalSignature,
com.novell.soa.af.impl.soap.SignaturePropertyArray
digitalSignaturePropertyArray, java.lang.String proxyUser)
```

Parameters

Parameter	Description
wid	The workId (activity Id).
action	The action to take (approve, deny, refuse).
items	The data items required by the workflow.
comment	The comment to add to the activity.
digitalSignature	The digital signature.
digitalSignaturePropertyArray	The digital signature property map.
proxyUser	The DN of the proxy user.

Example

```
     //
     // Initialize and start a provisioning request
     HashMap provMap = new HashMap();
     provMap.put(Helper.RECIPIENT, recipient);
     provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
     //
     // Start request
     // Calls method startProvisioningRequest on the provUtils
     // utility object which refers to a utility class that does not
     // ship with the Identity Manager User Application.
     String requestId = provUtils.startProvisioningRequest(provMap, null);
     sleep(5);
     //
    // Get the process id for this running process
    Process process = stub.getProcess(requestId);
    String processId = null;
    if(process != null)
        processId = process.getProcessId();


    T_Action action = T_Action.APPROVE;

    T_Logic logic = T_Logic.AND;

    T_WorkEntryOrder workEntryOrder = T_WorkEntryOrder.REQUEST_ID;

    T_WorkEntryQueryChoice [] workEntryqueryChoice =  new
T_WorkEntryQueryChoice[3];
    workEntryqueryChoice[0] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[0].setRecipient(recipient);
    workEntryqueryChoice[1] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[1].setRequestId(requestId);
    workEntryqueryChoice[2] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[2].setProcessId(processId);
    //
```

```
    //  work entry query
    T_WorkEntryQuery query = new T_WorkEntryQuery(logic, _workEntryOrder,
workEntryqueryChoice);
    //
    // Get all work entries (max records)
    WorkEntryArray workEntryArray = stub.getWorkEntries(query, -1);

    WorkEntry [] workEntry = workEntryArray.getWorkentry();

    if(workEntry != null

    {
         for(int wIndex = 0; wIndex < workEntry.length; wIndex++)
        {
            String workId = workEntry[wIndex].getId();
            //
            //
            LoggerUtils.sendToLogAndConsole("Forwarding : " +
workEntry[wIndex].getActivityName() + " work id: " + workId);
            //
            // Get the dataitem for this item of work
            DataItemArray dataItemArray = stub.getWork(workId);
            DataItem [] dataItem = dataItemArray.getDataitem();
            DataItemArray newDataItemArray = null;
            if(dataItem != null)
                // Call method replicateDataItemArray on the
                // provUtils utility object, which refers to a
                // utility class that does not ship with the
                // Identity Manager User Application.
                newDataItemArray =
provUtils.replicateDataItemArray(dataItem);
            else
                throw new TestProgramException("DataItem is null.");
            //
            // Claim request for recipient
            String comment = _action.toString() + " this request: " +
requestId + " for " + recipient;
            String digitalSignature =
DigitalSignatureUtils.getDigitalSignatureFromFile(IDigitalSignatureConstan
ts.MMACKENZIE_DIGITAL_SIGNATURE_FILENAME);
            String proxyUser =
ServiceUtils.getInstance().getLoginData().getUsername(LoginData.PROXY_TYPE
);

            stub.forwardAsProxyWithDigitalSignature(workId, _action,
newDataItemArray, comment, digitalSignature, null, proxyUser);
        }

    }
```

reassign

Used to reassign a task from one user to another.

Method Signature

```
void reassign(java.lang.String wid, java.lang.String addressee,
java.lang.String comment)
```

Parameters

Parameter	Description
wid	The Id of the activity to be reassigned.
addressee	The addressee of the activity.
comment	A comment about the action.

Example

```
       //
       // Initialize and start a provisioning request
       HashMap provMap = new HashMap();
       provMap.put(Helper.RECIPIENT, recipient);
       provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
       //
       // Start request
       // Calls method startProvisioningRequest on the provUtils
       // utility object which refers to a utility class that does not
       // ship with the Identity Manager User Application.
       String requestId = provUtils.startProvisioningRequest(provMap,
null);
       sleep(5);
       //
      // Get the process id for this running process
      Process process = stub.getProcess(requestId);
      if(process != null)
      {
            String processId = process.getProcessId();
            String initiator = process.getInitiator();
            //
            // Setup for the query
            HashMap map = new HashMap();
            map.put(Helper.REQUESTID, requestId);
            map.put(Helper.RECIPIENT, recipient);
            map.put(Helper.PROCESSID, processId);
            map.put(Helper.INITIATOR, initiator);
            WorkEntry [] workEntry =
workEntryUtils.getWorkEntriesUsingQuery(map, T_WorkEntryOrder.REQUEST_ID,
T_Logic.AND);

            if(workEntry == null)
                throw new TestProgramException("Work list is empty.");
            //
            // Reassign the work entry from recipient to the addressee
            //
```

```
            // Should only be one work item
            String reassignComment = null;
            String workId = workEntry[0].getId();
            if(workId != null)
            {
                //
                // Reassign work entry(s) to addressee
                reassignComment = "Reassigning work entry " + workId + "
from " + recipient + " to " + addressee;
                stub.reassign(workId, addressee, reassignComment);
                LoggerUtils.sendToLogAndConsole("Reassign work entry " +
workId + " from " + recipient + " to " + addressee);
            }
    }
```

getWorkEntries

Used to query the work entries (activities) and returns a list of `WorkEntry` objects that satisfy the query.

Method Signature

```
com.novell.soa.af.impl.soap.WorkEntryArray
getWorkEntries(com.novell.soa.af.impl.soap.T_WorkEntryQuery query, int
maxRecords)
```

Parameters

Parameter	Description
query	Used to specify the query used to retrieve the list of activities. The query has the following components: ◆ `choice` - the parameters used to filter the results. You can specify multiple parameters. The possible parameters are: Addressee - Possible values for this parameter are a DN or `self`. Use `self` if you want to retrieve work entries for the caller of the query, as identified by the authentication header of the SOAP header. ProcessId RequestId ActivityId Status (an integer) Owner Priority CreationTime (YYYY/MM/DD) ExpTime (YYYY/MM/DD) CompletionTime (YYYY/MM/DD) Recipient Initiator ProxyFor ◆ `logic` - AND or OR ◆ `order` - the order in which to sort the results. Possible values for `order` are: ACTIVITY_ID RECIPIENT PROVISIONING_TIME RESULT_TIME STATE STATUS REQUEST_ID MESSAGE
maxRecords	Used to specify maximum number of records to retrieve. A value of -1 returns unlimited records.

Example

```
    T_Action action = T_Action.APPROVE;

    T_Logic logic = T_Logic.AND;

    T_WorkEntryOrder workEntryOrder = T_WorkEntryOrder.REQUEST_ID;

    T_WorkEntryQueryChoice [] workEntryqueryChoice =  new
T_WorkEntryQueryChoice[3];
    workEntryqueryChoice[0] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[0].setRecipient(recipient);
    workEntryqueryChoice[1] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[1].setRequestId(requestId);
    workEntryqueryChoice[2] = new T_WorkEntryQueryChoice();
    workEntryqueryChoice[2].setProcessId(processId);
    //
    //  work entry query
    T_WorkEntryQuery query = new T_WorkEntryQuery(logic, _workEntryOrder,
workEntryqueryChoice);
    //
    // Get all work entries (max records)
    WorkEntryArray workEntryArray = stub.getWorkEntries(query, -1);

    WorkEntry [] workEntry = workEntryArray.getWorkentry();
```

getQuorumForWorkTask

Used to get information about the quorum for a workflow activity. A quorum must have actually been specified for the workflow activity by the workflow designer for this method to work.

Method Signature

```
com.novell.soa.af.impl.soap.Quorum getQuorumForWorkTask((java.lang.String
workId)
```

Example

```
     //

     // Note: Provisioning resource must contain a quorum in the flow for
this api method to work

    //
    // Action and Approval Types
    final int SELECTED_ACTION = 0; final int CLAIMED_SELECTED_ACTION = 0;
    T_Action [] action = {T_Action.APPROVE, T_Action.REFUSE,
T_Action.DENY};
    T_ApprovalStatus [] claimedAction = {T_ApprovalStatus.Approved,
T_ApprovalStatus.Retracted, T_ApprovalStatus.Denied};
    //
    // Get the process id for this running process
    Process process = stub.getProcess(requestId);
    String processId = null;
```

```
    if(process != null)
        processId = process.getProcessId();
    //
    // Setup for the query
    HashMap map = new HashMap();
    map.put(Helper.REQUESTID, requestId);
    map.put(Helper.RECIPIENT, recipient);
    map.put(Helper.PROCESSID, processId);
    map.put(Helper.INITIATOR, process.getInitiator() );
    WorkEntry [] workEntry =
workEntryUtils.getWorkEntriesUsingQuery(map,
T_WorkEntryOrder.REQUEST_ID, T_Logic.AND);

    Assert.assertNotNull("WorkEntry is null for recipient : " +
recipient + " with request id : " + requestId, workEntry);
    //
    //
    String workId = workEntry[0].getId();

    Quorum quorum = stub.getQuorumForWorkTask(workId);

    Assert.assertNotNull("Quorum for work task is null for recipient :
" + recipient + " with request id : " + requestId, quorum);
    //

    // Extract some data
    int approvalCondition = quorum.getApprovalCondition();
    int status = quorum.getStatus();
    int approveCount = quorum.getApproveCount();
    int participantCount = quorum.getParticipantCount();
    int refuseCount = quorum.getRefuseCount();
```

resetPriorityForWorkTask

Used to reset the priority for a task. You should only use this method on provisioning requests that have a single approval branch.

Method Signature

```
void resetPriorityForWorkTask(java.lang.String workId, int priority,
java.lang.String comment)
```

Parameters

Parameter	Description
workId	The Id of the activity.
priority	The priority to set for the activity.
comment	A comment about the action.

Example

```
// Calls method getProvisioningResourceNameForRecipient
// on the provUtils utility object, which refers to a utility class
// that does not ship with the Identity Manager User Application.
String requestNameToStart =
provUtils.getProvisioningResourceNameForRecipient(recipient, "Enable
Active Directory Account");
    Map map = MapUtils.createAndSetMap(new Object[] {
            Helper.RECIPIENT, recipient,
            IProvisioningConstants.PROVISIONING_REQUEST_TO_START,
requestNameToStart});
    //
    // Try and start the provisioning request
     String requestId =
provWrapper.startProvisioningRequest(recipient, requestNameToStart);
    RationalTestScript.sleep(5);
    //
    // Get the process id for this running process
    Process process = stub.getProcess(requestId);
    if(process != null)
    {
        //
        // Setup for the query
        HashMap map = new HashMap();
        map.put(Helper.REQUESTID, requestId);
        map.put(Helper.RECIPIENT, recipient);
        map.put(Helper.PROCESSID, process.getProcessId());
        map.put(Helper.INITIATOR, process.getInitiator());
        WorkEntry [] workEntry =
workEntryUtils.getWorkEntriesUsingQuery(map,
T_WorkEntryOrder.REQUEST_ID, T_Logic.AND);
        //
        // Now reset the priority for this work item.
        String workId = workEntry[0].getId();
        String comment = "Resetting priority for this work item.";
        int priority = 0;
        stub.resetPriorityForWorkTask(workId, priority, comment);
}
```

Comments

This section provides reference information for each Comments method.

getCommentsByType

Used to get workflow comments that are of a specific type (for example, user, system).

Method Signature

```
com.novell.soa.af.impl.soap.CommentArray
getCommentsByType(java.lang.String requestId,
com.novell.soa.af.impl.soap.T_CommentType type)
```

Parameters

Parameter	Description
requestId	The process identifier.
type	The comment type (USER or SYSTEM)

Example

```
       //
       // Initialize and start a provisioning request
       HashMap provMap = new HashMap();
       provMap.put(Helper.RECIPIENT, recipient);
       provMap.put(I"Provisioning_Request_To_Start_Key", "Enable
Active Directory Account (Mgr Approve-No Timeout)");
       //
       // Start request
       // Calls method startProvisioningRequest on the provUtils
       // utility object which refers to a utility class that does not
       // ship with the Identity Manager User Application.
       String requestId = provUtils.startProvisioningRequest(provMap,
null);
      sleep(5);
      //
      // Get the comments by type : either User or System
      T_CommentType [] commentTypes = {T_CommentType.User,
T_CommentType.System};

     for(int types = 0; types < commentTypes.length; types++)
    {
        CommentArray commentArray = stub.getCommentsByType(requestId,
commentTypes[types]);
        Comment [] comments = commentArray.getComment();
        if(comments != null)
        {
            for(int index = 0; index < comments.length; index++)
            {
                LoggerUtils.sendToLogAndConsole(" \nComment Type = " +
commentTypes[types].getValue() + "\n" +
                            "Activity Id: " +
comments[index].getActivityId() + "\n" +
                            "Comment : " + comments[index].getComment()
+ "\n" +
                            "User : " + comments[index].getUser() + "\n"
+
                            "System comment : " +
comments[index].getSystemComment() + "\n" +
                            "Time stamp : " +
comments[index].getTimestamp().getTime().toString() );
           }
        }
    }
```

getCommentsByActivity

Used to get the comments for a specific activity.

Method Signature

```
com.novell.soa.af.impl.soap.CommentArray
getCommentsByActivity(java.lang.String requestId, java.lang.String aid)
```

Parameters

Parameter	Description
requestId	The process identifier.
aid	The activity identifier.

Example

```
       //
       // Initialize and start a provisioning request
       HashMap provMap = new HashMap();
       provMap.put(Helper.RECIPIENT, recipient);
       provMap.put(I"Provisioning_Request_To_Start_Key", "Enable
Active Directory Account (Mgr Approve-No Timeout)");
       //
       // Start request
       // Calls method startProvisioningRequest on the provUtils
       // utility object which refers to a utility class that does not
       // ship with the Identity Manager User Application.
       String requestId = provUtils.startProvisioningRequest(provMap,
null);
      sleep(5);
      //
      // Get the process id for this running process
      Process process = stub.getProcess(requestId);
      if(process != null)
      {
            String processId = process.getProcessId();
            String initiator = process.getInitiator();
            //
            // Setup for the query
            HashMap map = new HashMap();
            map.put(Helper.REQUESTID, requestId);
            map.put(Helper.RECIPIENT, recipient);
            map.put(Helper.PROCESSID, processId);
            map.put(Helper.INITIATOR, initiator);
            WorkEntry [] workEntry =
workEntryUtils.getWorkEntriesUsingQuery(map,
```

```
T_WorkEntryOrder.REQUEST_ID, T_Logic.AND);
            //
            // Get the activity id associated with the item of work
            String activityId = workEntry[0].getActivityId();
            //
            // Get the comments based on activity
            if(activityId != null)
            {
                CommentArray commentArray =
stub.getCommentsByActivity(requestId, activityId);
                Comment [] comments = commentArray.getComment();
            }

    }
```

getCommentsByUser

Used to get the comments made by a specific user.

Method Signature

```
com.novell.soa.af.impl.soap.CommentArray
getCommentsByUser(java.lang.String requestId, java.lang.String user)
```

Parameters

Parameter	Description
requestId	The process identifier.
user	The the DN of the user (recipient) who created the comments

Example

```
       //
       // Initialize and start a provisioning request
       HashMap provMap = new HashMap();
       provMap.put(Helper.RECIPIENT, recipient);
       provMap.put(I"Provisioning_Request_To_Start_Key", "Enable
Active Directory Account (Mgr Approve-No Timeout)");
       //
       // Start request
       // Calls method startProvisioningRequest on the provUtils
       // utility object which refers to a utility class that does not
       // ship with the Identity Manager User Application.
       String requestId = provUtils.startProvisioningRequest(provMap,\
null);
      sleep(5);
     //
      // Get the comments by recipient (should be the same as user)
      CommentArray commentArray = stub.getCommentsByUser(requestId,
recipient);
      Comment [] comments = commentArray.getComment();
```

getCommentsByCreationTime

Used to get comments made at a specific time.

Method Signature

```
com.novell.soa.af.impl.soap.CommentArray
getCommentsByCreationTime(java.lang.String requestId, long time,
com.novell.soa.af.impl.soap.T_Operator op)
```

Parameters

Parameter	Description
requestId	The process identifier.
time	The time stamp.
op	The query operator to use. Possible values for operator are: EQ - equals LT - less than LE - less than or equal to GT - greater than GE - greater than or equal to

Example

```
       //
       // Initialize and start a provisioning request
       HashMap provMap = new HashMap();
       provMap.put(Helper.RECIPIENT, recipient);
       provMap.put(I"Provisioning_Request_To_Start_Key", "Enable
Active Directory Account (Mgr Approve-No Timeout)");
       //
       // Start request
       // Calls method startProvisioningRequest on the provUtils
       // utility object which refers to a utility class that does not
       // ship with the Identity Manager User Application.
       String requestId = provUtils.startProvisioningRequest(provMap,
null);
      sleep(5);
      //
      // Get comments by creation time for the provisioning request
started above.
      long currentTime = System.currentTimeMillis();
      LoggerUtils.sendToLogAndConsole("-->Current date = " + new
java.util.Date(currentTime).toString() );
      //
      //
      T_Operator operator = T_Operator.GT;
      CommentArray commentArray =
stub.getCommentsByCreationTime(requestId, currentTime, operator);
      Comment [] comments = commentArray.getComment();
```

addComment

Used to add a comment to a workflow activity.

Method Signature

```
void addComment(java.lang.String workId, java.lang.String comment)
```

Parameters

Parameter	Description
workId	The activity identifier (UUID).
comment	A comment about the activity.

Example

```
       //
       // Initialize and start a provisioning request
       HashMap provMap = new HashMap();
       provMap.put(Helper.RECIPIENT, recipient);
       provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
       //
       // Start request

       // Calls method startProvisioningRequest on the provUtils
       // utility object which refers to a utility class that does not
       // ship with the Identity Manager User Application.

       String requestId = provUtils.startProvisioningRequest(provMap,
null);
      sleep(5);
      //
      // Setup for the query
      HashMap map = new HashMap();
      map.put(Helper.REQUESTID, requestId);
      map.put(Helper.RECIPIENT, recipient);
     WorkEntry [] workEntry = workEntryUtils.getWorkEntriesUsingQuery(map,
T_WorkEntryOrder.REQUEST_ID, T_Logic.AND);
      //
      // Add comment to the work entry
      String workId = workEntry[0].getId();
      String processId = workEntry[0].getProcessId();
      String addComment = "Test comment for work id " + workId;
      stub.addComment(workId, addComment);
      sleep(2);
```

getComments

Used to get comments from a workflow.

Method Signature

```
com.novell.soa.af.impl.soap.CommentArray getComments(java.lang.String
workId, int maxRecords)
```

Parameters

Parameter	Description
workId	The activity Id (UUID).
maxRecords	An integer specifying the maximum number of records to retrieve.

Example

```
      //
      // Setup for the query
      HashMap map = new HashMap();
      map.put(Helper.RECIPIENT, addressee);
      WorkEntry [] workEntry =
workEntryUtils.getWorkEntriesUsingQuery(map,
T_WorkEntryOrder.ADDRESSEE, T_Logic.OR);
      //
      // Get all the comment records for this workId
      int maxRecords = -1;
      CommentArray commentArray = stub.getComments(workId, maxRecords);
      Comment [] comment = commentArray.getComment();
```

Configuration

This section provides reference information for each Configuration method.

setCompletedProcessTimeout

Used to set the timeout for completed processes. Processes that were completed more than timeout days ago are removed from the system. The default value is 120 days. The valid range is 0 days to 365 days.

Method Signature

```
void setCompletedProcessTimeout(int time)
```

Example

```
accessConfigurationSettings(SET_COMPLETED_PROCESS_TIMEOUT, new
Integer(212) );
```

setEngineConfiguration

Used to set workflow engine configuration parameters.

Method Signature

```
void  setEngineConfiguration(com.novell.soa.af.impl.soap.Configuration
config)
```

Parameters

Parameter	Description
minPoolSize	The minumum thread pool size.
maxnPoolSize	The maximum thread pool size.
initialPoolSize	The initial thread pool size.

Parameter	Description
keepAliveTime	Thread pool keep live time.
pendingInterval	The cluster synchronization time.
cleanupInterval	The interval between purging processes from databases.
retryQueueInterval	The interval between retrying failed processes.
maxShutdownTime	The maximum time to let threads complete work before engine shutdown.
userActivityTimeout	The default user activity timeout.
completedProcessTimeout	The default completed process timeout.
webServiceActivityTimeout	The default Web service activity timeout.
emailNotification	Turns email notification on or off.
processCacheInitialCapacity	The process cache initial capacity.
processCacheMaxCapacity	The process cache maximum capacity.
processCacheLoadFactor	The process cache load factor.
heartbeatInterval	The heartbeat interval.
heartbeatFactor	The heartbeat factor.

Example

```
accessConfigurationSettings(SET_ENGINE_CONFIGURATION, new Integer(313) );
```

getCompletedProcessTimeout

Used to get the timeout for completed processes.

Method Signature

```
int getCompletedProcessTimeout()
```

Example

```
accessConfigurationSettings(GET_COMPLETED_PROCESS_TIMEOUT, new
Integer(121) );
```

setEmailNotifications

Used to globally enable or disable email notifications.

Method Signature

```
void setEmailNotifications(boolean enable)
```

Parameters

Parameter	Description
enable	Email notifications are enabled if true; otherwise they are disabled.

Example

```
accessConfigurationSettings(SET_EMAIL_NOTIFICATIONS, new Boolean(false) );
```

clearNIMCaches

Clear the NetIQ Integration Manager (previously named exteNd Composer) caches.

Method Signature

```
void clearNIMCaches()
```

Example

```
accessConfigurationSettings(CLEAR_NIM_CACHES, new Object() );
```

setWebServiceActivityTimeout

Used to set the timeout for Web service activities. The default value is 50 minutes. The valid range is 1 minute to 7 days.

Method Signature

```
void setWebServiceActivityTimeout(int time)
```

Parameters

Parameter	Description
time	The timeout value in minutes.

Example

```
accessConfigurationSettings(SET_WEBSERVICE_ACTIVITY_TIMEOUT, new
Integer(767) );
```

getUserActivityTimeout

Used to get the timeout for user-facing activities.

Method Signature

```
int getUserActivityTimeout()
```

Example

```
accessConfigurationSettings(GET_USER_ACTIVITY_TIMEOUT, new Integer(3767)
);
```

getEmailNotifications

Used to determine if global email notifications are enabled or disabled.

Method Signature

```
boolean getEmailNotifications()
```

Example

```
accessConfigurationSettings(GET_EMAIL_NOTIFICATIONS, new Boolean(true) );
```

setUserActivityTimeout

Used to set the timeout for user-facing activities. The default value is no timeout (a value of zero). The valid range is 1 hour to 365 days.

Method Signature

```
void setUserActivityTimeout(int time)
```

Parameters

Parameter	Description
time	The timeout value in hours.

Example

```
accessConfigurationSettings(SET_USER_ACTIVITY_TIMEOUT, new Integer(1767)
);
```

getEngineConfiguration

Used to get the workflow engine configuration parameters.

Method Signature

```
com.novell.soa.af.impl.soap.Configuration getEngineConfiguration()
```

Example

```
accessConfigurationSettings(GET_ENGINE_CONFIGURATION, new Integer(141) );
```

getWebServiceActivityTimeout

Used to get the timeout for Web service activities.

Method Signature

```
int getWebServiceActivityTimeout()
```

Example

```
accessConfigurationSettings(GET_WEBSERVICE_ACTIVITY_TIMEOUT, new
Integer(808) );
```

Miscellaneous

This section provides reference information for each Miscellaneous method.

getGraph

Used to get a JPG image of the workflow. The Graphviz program must be installed on the computer where the application server and the IDM User Application is running. For more information about Graphviz, see Graphviz (http://www.graphviz.org).

Method Signature

```
byte[] getGraph(java.lang.String processId)
```

Parameters

Parameters	Description
processId	The request Id.

Example

```
 //
     // Initialize and start a provisioning request
     HashMap provMap = new HashMap();
     provMap.put(Helper.RECIPIENT, recipient);
     provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
     //
     // Start request
     // Calls method startProvisioningRequest on the provUtils
     // utility object which refers to a utility class that does not
     // ship with the Identity Manager User Application.
     String requestId = provUtils.startProvisioningRequest(provMap,
null);
     sleep(5);
     //

     //
```

```
     Process process = stub.getProcess(requestId);
     if(process != null)
    {
        byte [] graph = null;
        if( (recipient.compareTo(process.getRecipient()) == 0) &&
(requestId.compareTo(process.getRequestId()) == 0) )
        {
            graph = stub.getGraph(process.getProcessId() );
        }
        //
        // Do assert
        Assert.assertNotNull("Graph is null.", graph);
    }
```

getFlowDefinition

Used to get the XML for a provisioning request.

Method Signature

```
java.lang.String getFlowDefinition(java.lang.String processId)
```

Parameters

Parameters	Description
processId	The request Id.

Example

```
 //
     // Initialize and start a provisioning request
     HashMap provMap = new HashMap();
     provMap.put(Helper.RECIPIENT, recipient);
     provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
     //
     // Start request
     // Calls method startProvisioningRequest on the provUtils
     // utility object which refers to a utility class that does not
     // ship with the Identity Manager User Application.
     String requestId = provUtils.startProvisioningRequest(provMap, null);
     sleep(5);
     //

     //

     Process process = stub.getProcess(requestId);
```

```
     if(process != null)
    {
        String XMLFlowDefinition = null;
        if( (recipient.compareTo(process.getRecipient()) == 0) &&
(requestId.compareTo(process.getRequestId()) == 0) )
        {
            XMLFlowDefinition =
stub.getFlowDefinition(process.getProcessId() );
        }
        //
        // Do assert
        Assert.assertNotNull("Flow Definition is null.",
XMLFlowDefinition);
    }
```

getFormDefinition

Used to get the XML for a form for a provisioning request.

Method Signature

```
java.lang.String getFormDefinition(java.lang.String processId)
```

Parameters

Parameters	Description
processId	The request Id.

Example

```
 //
     // Initialize and start a provisioning request
     HashMap provMap = new HashMap();
     provMap.put(Helper.RECIPIENT, recipient);
     provMap.put(I"Provisioning_Request_To_Start_Key", "Enable Active
Directory Account (Mgr Approve-No Timeout)");
     //
     // Start request
     // Calls method startProvisioningRequest on the provUtils
     // utility object which refers to a utility class that does not
     // ship with the Identity Manager User Application.
     String requestId = provUtils.startProvisioningRequest(provMap, null);
     sleep(5);
     //

     //

     Process process = stub.getProcess(requestId);
     if(process != null)
```

```
    {

         String XMLFormDefinition = null;
         if( (recipient.compareTo(process.getRecipient()) == 0) &&
(requestId.compareTo(process.getRequestId()) == 0) )
        {
            XMLFormDefinition =
stub.getFormDefinition(process.getProcessId() );
        }
        //
        // Do assert
        Assert.assertNotNull("Form Definition is null.",
XMLFormDefinition);
    }
```

getVersion

Used to get the version of the workflow system.

Method Signature

```
com.novell.soa.af.impl.soap.T_Version getVersion()
```

Example

```
 StringBuffer result = new StringBuffer();

     T_Version version = stub.getVersion();
     if (version != null)
    {
        result.append(" Major = " + version.getMajor() );
        result.append(" Minor = " + version.getMinor() );
        result.append(" Revision = " + version.getRevision() );

        System.out.println("Version Information " + result.toString());
    }
```

Cluster

This section provides reference information for each Cluster method.

getEngineState

Used to get the `IEngineState` for a workflow engine, specified by engine Id.

Method Signature

```
com.novell.soa.af.impl.soap.EngineState getEngineState(java.lang.String
engineId)
```

Parameters

Parameter	Description
engineId	The Id of the workfow engine.

Example

```
 EngineStateArray engineStateArray = stub.getClusterState();
    EngineState [] engineState = engineStateArray.getEngineStates();
    if(engineState != null)
    {
        LoggerUtils.sendToLogAndConsole("EngineCount in cluster:" +
engineState.length);
        for(int index = 0; index < engineState.length; index++)
        {
            EngineState engine =
stub.getEngineState(engineState[index].getEngineId() );
            LoggerUtils.sendToLogAndConsole(
                "Engine Id: " + engine.getEngineId() + "\n" +
                "Engine status: " + engine.getEngineStatus() + "\n" +
                "Value of engine status: " +
engine.getValueOfEngineStatus() + "\n" +
                "Heartbeat: " + ( (engine.getHeartbeat() != null) ?
engine.getHeartbeat().getTime().toString() : "null") + "\n" +
                "Shutdown time: " + ((engine.getShutdownTime()!= null)
? engine.getShutdownTime().getTime().toString() : "null") + "\n" +
                "Start time: " + ((engine.getStartTime() != null) ?
engine.getStartTime().getTime().toString() : "null") );
        }
    }
```

reassignAllProcesses

Used to reassign all processes from the source engine to a list of target engines.

Method Signature

```
int reassignAllProcesses(java.lang.String sourceEngineId,
com.novell.soa.af.impl.soap.StringArray targetEngineIds)
```

Parameters

Parameter	Description
sourceEngineId	The Id of the source workflow engine.
targetEngineIds	The Ids of the target workflow engines.

getEngineState

Used to get a list that contains an IEngineState object for each engine in the cluster.

Method Signature

```
public com.novell.soa.af.impl.soap.EngineState
getEngineState(java.lang.String engineId)
```

Parameters

Parameter	Description
engineId	The Id of the workfow engine.

Example

```
 EngineStateArray engineStateArray = stub.getClusterState();
    EngineState [] engineState = engineStateArray.getEngineStates();
    if(engineState != null)
    {
        LoggerUtils.sendToLogAndConsole("EngineCount in cluster:" +
engineState.length);
        for(int index = 0; index < engineState.length; index++)
        {
            EngineState engine =
stub.getEngineState(engineState[index].getEngineId() );
            LoggerUtils.sendToLogAndConsole(
                "Engine Id: " + engine.getEngineId() + "\n" +
                "Engine status: " + engine.getEngineStatus() + "\n" +
                "Value of engine status: " +
engine.getValueOfEngineStatus() + "\n" +
                "Heartbeat: " + ( (engine.getHeartbeat() != null) ?
engine.getHeartbeat().getTime().toString() : "null") + "\n" +
                "Shutdown time: " + ((engine.getShutdownTime()!= null)
? engine.getShutdownTime().getTime().toString() : "null") + "\n" +
                "Start time: " + ((engine.getStartTime() != null) ?
engine.getStartTime().getTime().toString() : "null") );
        }
    }
```

reassignPercentageProcesses

Used to reassign a percentage of processes from the source engine to the target engine.

Method Signature

```
int reassignPercentageProcesses(int percent, java.lang.String
sourceEngineId, java.lang.String targetEngineId)
```

Parameters

Parameter	Description
percent	An integer representing the percentage of processes to be reassigned.
sourceEngineId	The Id of the source workflow engine.
targetEngineIds	The Id of the target workflow engine.

reassignProcesses

Used to reassign one or more processes from the source engine to the target engine.

Method Signature

```
int reassignProcesses(com.novell.soa.af.impl.soap.StringArray requestIds,
java.lang.String sourceEngineId, java.lang.String targetEngineId)
```

Parameters

Parameter	Description
requestIds	A list of requestIds of the processes to be reassigned.
sourceEngineId	The Id of the source workflow engine.
targetEngineIds	The Id of the target workflow engine.

removeEngine

Used to remove an engine from the cluster state table. The engine must be in the SHUTDOWN or TIMEDOUT state.

Method Signature

```
void removeEngine(java.lang.String engineId)
```

Parameters

Parameter	Description
engineId	The Id of the workflow engine to be removed.

26 Metrics Web Service

This section describes the Metrics Web Service, which provides metrics for provisioning workflows.

About the Metrics Web Service

The workflow engine includes a Web Service for gathering workflow metrics. The addition of the Metrics Web Service to the workflow engine lets you monitor an approval flow process. In addition, it provides indicators the business manager can use to modify the process for optimal performance.

The metrics are based on traditional business process flow management principles, which emphasize the need for metrics to be actionable. This ensures that the metrics provided match what an operations manager usually looks for when analyzing and optimizing business flows. Therefore, the metrics identify bottlenecks and provide other capacity indicators. The Metrics Web Service allows you to narrow down the metrics to a common and established set of data, instead of trying to anticipate the myriad of metrics and reports that can be created for a business process flow.

When working with the Metrics Web Service, you should keep in mind that the service is not intended to be an all-purpose metrics system:

- The Metrics Web Service is not a reporting tool or reporting engine. Consequently it does not use a complex query language.
- The Metrics Web Service is not designed as an all-purpose performance management system. This helps to limit the impact of the needed queries against the live system being monitored.

Operations management stresses three key internal process performance measures that together capture the essence of process flow. These three measures can serve as leading indicators of customer satisfaction: flow time, flow rate, and inventory.

With these measures, an operations manager can answer the following questions:

- On average, how much time does a provisioning request spend within the process boundaries? (Flow time)
- On average, how many provisioning requests pass through the process per unit of time? (Flow rate)
- On average, how many provisioning requests are within the process boundaries at any point in time? (Inventory)

These three measures are related by Little's law:

```
Inventory=Flow Rate*Flow Time
```

Web Service Semantics

The following semantics apply to the use of the Metrics Web Service:

- Activities in the Metrics Web Service refer only to user-facing activities (Approval Activities). Negligible running time and the impossibility of controlling the other activities make collecting metrics for these inappropriate.
- The Metrics Web Service distinguishes between Working Days and Calendar Days. Calendar Days refer to all days between two dates. Working Days refer only to working days between two dates. Since working days may be specified differently in different environments, all Working Days methods return a raw data set that can be used to compute what is appropriate. If no such detail is required, the Calendar Days method will readily return the appropriate metric.

Accessing the Test Page

The Metrics Web Service endpoint can be accessed at the following URL:

`http://server:port/warcontext/metrics/service`

You can also access the SOAP endpoint by going to the **Administration** within the User Application. To do this, you need to select the **Application Configuration** tab, then select **Web Services** from the left-navigation menu. After selecting **Web Services**, pick the Web Service endpoint you want from the list.

WARNING: The test page is disabled by default. Since some of the methods allow data to be updated, the test page presents a potential security vulnerability and should not be allowed in a production environment. For details on enabling the test page, see the instructions provided for the Role Service in Enabling the Test Page.

Web Service Methods Grouped by Security Permissions

The service is secured using Basic Authentication. Therefore, you should use SSL to connect to the service. The service uses the same security layer as the User Application and consequently not all service operations are allowed to all users. Only a Provisioning Administrator will have unconditional access to all the methods. On the other hand team managers will only have access to metrics that pertain to their team and team members.

Hence the Metrics Web Service operations are divided into 3 categories according to role and security permissions:

- Team manager operations
- Provisioning Application Administrator operations
- Utility operations

Team Metrics

Team managers can only retrieve metrics on a team for which they are managers. These are the methods are available to team managers:

Table 26-1 *Team Metrics Methods*

Method	Description
getClaimedFlowTimeCalendarDays	Returns the average time in hours the provisioning request was claimed for within the specified time interval
getClaimedFlowTimeWorkingDays	Returns the result set required to compute the average time the provisioning request was claimed for the specified time interval
getToClaimedFlowTimeCalendarDays	Returns the average time in hours it took the provisioning request to be claimed from the moment it was available to addressees
getToClaimedFlowTimeWorkingDays	Returns the average time it took the provisioning request to be claimed from the moment it was available to addresses, within the specified time interval
getClaimedInventory	Returns the average number of provisioning requests claimed within the specified interval
getClaimedThroughputWorkingDays	Returns the average number of provisioning requests claimed within the specified interval
getTeamLongestRunning	Returns a result set of the longest running request in seconds for which members of the team acted as addressees
getTeamFlowHistory	Returns a result set of the activity outcomes, addressee and addressee messages for the specified list of provisioning requests
getTeamHistoryForInitiators	Returns a result set of the provisioning request and their status for which members of the team acted as initiators
getTeamHistoryForRecipients	Returns a result set of the provisioning request and their status for which members of the team acted as recipients
getTeamRunningTime	Returns the average time in seconds the specified provisioning requests have been running
getTeamDecisionCount	Returns the number of decisions the team made as addressees for the specified provisioning request
getTeamInitiatedCount	Returns the number of provisioning requests initiated by the team
getTeamRecipientCount	Returns the provisioning requests for which a member of the team acts as a recipient

Provisioning Administrator Metrics

This role is unrestricted and may perform any of the service's operations. These are the methods that are only available to Provisioning Administrators.

Table 26-2 *Provisioning Administrator Metrics Methods*

Method	Description
getActivityFlowTimeCalendarDays	Returns the average time in hours the user activity took to complete
getActivityFlowTimeWorkingDays	Returns the result set required to compute the average time the user activity took to complete
getActivityInventory	Returns the average number of provisioning requests at any one time for the specified user activity
getActivityThroughputCalendarDays	Returns the average number of provisioning requests per hours that exited the specified user activity within the specified time interval
getActivityThroughputWorkingDays	Returns the result set required to compute average time it takes a provisioning request to complete for the specified time interval
getFlowTimeCalendarDays	Returns average time in hours it takes a provisioning request to complete for the specified time interval
getFlowTimeWorkingDays	Returns the result set required to compute average time it takes a provisioning request to complete for the specified time interval
getInventory	Returns the average number of provisioning requests in the system at any one time for the specified time interval
getLongestClaimed	Returns a result set of the provisioning requests that have been claimed but not acted upon (time in seconds)
getLongestRunning	Returns a result set of the longest running provisioning requests (time in seconds)
getFlowCount	Returns the number of provisioning requests
getFlowHistory	Returns a result set of the activity outcomes, addressee and addressee messages for the specified list of provisioning requests
getFlowHistoryForInitiators	Returns the list of provisioning requests and their status for the specified initiators
getFlowHistoryForRecipients	Returns the list of provisioning requests and their status for the specified recipients
getRunningTime	Returns the average running time in seconds for the provisioning requests that are currently running

Method	Description
getThroughputCalendarDays	Returns the average number of provisioning requests per hour that completed within the specified interval
getThroughputWorkingDays	Returns the result set required to compute the average number per hour of provisioning requests that completed within the specified interval

Utility Operations

Both team managers and administrators may perform these operations:

Table 26-3 *Utility Operations*

Method	Description
getVersion	Returns the server version of the Web service. This should always used to ensure version matching between client and server code.
getAllProvisioningFlows	Returns the list of provisioning flows that the logged in user can see
getUserActivityOnlyFlow	Returns a GraphViz DOT (http://www.graphviz.org/) representation of the provisioning workflow
getTeams	Returns the list of teams the logged in user manages
getTeamMembers	Returns the list of team members for the specified team

Specifying Filters

As mentioned above, the Metrics Webservice does not use a complex query language. Instead filters can be use to narrow results by criteria such as date ranges or approval statuses.

These are the filters you can specify (see type FilterConstants in service's WSDL):

Table 26-4 *Filters for Narrowing Metric Results*

Filter	Description
KEY_ACTIVITY_ID	A User Activity Id as defined in the provisioning request definition

Filter	Description
KEY_APPROVAL_STATUS	The approval status for the provisioning request. Possible values are: ◆ ApprovalStatusProcessing ◆ ApprovalStatusDenied ◆ ApprovalStatusRefused ◆ ApprovalStatusApproved ◆ ApprovalStatusRetract ◆ ApprovalStatusError
KEY_ENTITLEMENT_STATE	The state of the entitlement associated with the provisioning request. Possible value are: ◆ EntitlementUnknown ◆ EntitlementGranted ◆ EntitlementRevoked
KEY_ENTITLEMENT_STATUS	The status of the entitlement associated with the provisioning request. Possible values are: ◆ EntitlementSuccess ◆ EntitlementWarning ◆ EntitlementError ◆ EntitlementFatal
KEY_INITIATOR	The user DN of the workflow initiator
KEY_L_COMPLETION_TIME	The date indicating the start of the interval for workflow completion
KEY_S_COMPLETION_TIME	The date indicating the end of the interval for workflow completion
KEY_L_ENTITLEMENT_TIME	The date indicating the start of the interval for entitlement time
KEY_S_ENTITLEMENT_TIME	The date indicating the end of the interval for entitlement time
KEY_S_START_TIME	The date indicating the start of the interval for workflow start
KEY_L_START_TIME	The date indicating the end of the interval for workflow start
KEY_PROCESS_ID	The DN of the provisioning request

Filter	Description
KEY_PROCESS_STATUS	The status of the provisioning request. Possible values are: ◆ ProcessStatusRunning ◆ ProcessStatusStopped ◆ ProcessStatusTerminated ◆ ProcessStatusCompleted
KEY_PROCESS_VERSION	The process version associated with the workflow version
KEY_RECIPIENT	The user DN of the workflow recipient
KEY_REQUEST_ID	The unique id associated with the workflow instance

Here is a Java example. Note that your code will obviously differ depending on the platform you use for your Web Service client:

```
HashMap map=new HashMap();
map.put(MetricsFilter.KEY_PROCESS_STATUS,
MetricsFilter.ProcessStatusRunning);
double flowtime = metrics.getFlowTimeCalendarDays(processId,
  processVer, activity, 5, calendar1.getTime(),
  calendar2.getTime(), MetricsFilter.ACTIVITY_CLAIMED,
  MetricsFilter.ACTIVITY_FORWARDED, map);
...
```

Please consult the WebService WSDL for more information:

`http://server:port/warcontext/metrics/service?WSDL`

Generating the Stub Classes

Before using the Web Service, you need to use the WSSDK tool or another SOAP tool kit to generate the stub classes. To allow your code to find the stub classes, you also need to add the JAR that contains the stub classes to your classpath.

If you want to use the NetIQ WSSDK tool, you can generate the client stubs by extracting the WSDL and running the wsdl2java utility. For example, you could run this command to generate the stubs in a package called com.novell.soa.af.metrics.soap.impl:

```
"C:\Program Files\Java\jdk1.6.0_31\bin\java" -cp "../lib/wssdk.jar;../lib/
jaxrpc-api.jar";"../lib/mail.jar";"../lib/activation.jar";"c:\Program
Files\Java\jdk1.6.0_31\lib\tools.jar";
com.novell.soa.ws.impl.tools.wsdl2java.Main -verbose -ds gensrc -d C:\ -
noskel -notie -genclient -keep -package
com.novell.soa.af.metrics.soap.impl -javadoc metrics.wsdl
```

You can change the wsdl2java parameters to suit your requirements.

Obtaining the Remote Interface

Before you can begin calling methods on the Metrics Web Service, you need to have a reference to the remote interface.

The code below shows how to obtain the remote interface.

```
import java.util.Locale;
import java.util.Properties;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.xml.rpc.Stub;
import com.novell.qa.soap.common.util.LoggerUtils;
import com.novell.qa.soap.common.util.LoginData;
import com.novell.qa.soap.common.util.ServiceUtils;
import com.novell.soa.af.ClusterException;
import com.novell.soa.af.impl.soap.Provisioning;
import com.novell.soa.af.impl.soap.ProvisioningService;
import com.novell.test.automator.framework.TestProgramException;
import com.rational.test.ft.script.RationalTestScript;
import com.novell.soa.af.metrics.soap.MetricsClientHelper;
import com.novell.soa.af.metrics.soap.MetricsStubWrapper;
import com.novell.soa.af.metrics.soap.impl.MetricsService;
import com.novell.soa.af.metrics.soap.impl.MetricsServiceException;
import com.novell.soa.af.metrics.soap.impl.IRemoteMetrics;


/**
* Method to obtain the remote interface to the Metrics endpoint
* @param _url
* @param _username
* @param _password
* @return IRemoteMetrics interface
* @throws Exception
*/
private IRemoteMetrics getStub(String _url, String _username, String
_password) throws Exception

{
   Properties properties = new Properties();
   properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jnp.interfaces.NamingContextFactory");

    String lookup =
```

```
"xmlrpc:soap:com.novell.soa.af.metrics.soap.impl.MetricsService";

    InitialContext ctx = new InitialContext();
    MetricsService svc = (MetricsService) ctx.lookup(lookup);

    Stub stub = (Stub)svc.getIRemoteMetricsPort();

    stub._setProperty(Stub.USERNAME_PROPERTY, _username);
    stub._setProperty(Stub.PASSWORD_PROPERTY, _password);
    stub._setProperty(Stub.SESSION_MAINTAIN_PROPERTY, Boolean.TRUE);
    stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, _url);

    return (IRemoteMetrics) stub;
}
```

Here's the code to call the method defined above:

```
IRemoteMetrics stub = null;
        try
        {
            //
            // Get the stub
            String url = m_loginData.getURL();
            stub = getStub(url,_username,_password);
        }
        catch(Exception e)
        {
            String msg = e.getMessage();
            LoggerUtils.logError(msg);
            throw new TestProgramException(msg);
        }
        return stub;
```

In order for this code to work, the URL passed to the getStub() method would need to point to the SOAP endpoint, as shown below:

```
http://myserver:8080/IDMProv/metrics/service
```

The user name needs to be a fully qualified DN such as the following:

```
"cn=admin,ou=idmsample,o=netiq"
```

Metrics Configuration Settings

The Metrics Web Service impact on the live system is limited by 4 settings that may be modified in the `IDMfw.jar/WorkflowService-conf/config.xml` file:

Table 26-5 *Metrics Configuration Settings*

Key in config.xml	Description
<key>Metrics/TimeRequiredBetweenClientRequests</key>	Required time between client requests in ms (default is 250 ms)
<key>Metrics/MaxClients</key>	Maximum number of concurrent client sessions (default is 10)
<key>Metrics/MaxRows</key>	Maximum number of rows any query can return
<key>Metrics/MaxTeamMembers</key>	Maximum Number of Team Members
<key>Metrics/SecondsToAnythingDivider</key>	The divider used in all throughput computations (default 3600). Original values are in seconds so all throughputs are consequently per hour.

When the limit has been reached for any of these settings a Web Service fault is generated indicating the problem. In addition, for settings 1 and 2, the fault includes an error code.

- If the fault is caused by a TimeRequiredBetweenClientRequests error, the error code is 100.
- If the fault is caused by a MaxClients errors, the error code is 200.
- If the fault is caused by a closed client connection error, the error code is 300.

Client consumers of the Metrics Web Service will have to include in their code provisions for retrying a request. Here is a simple Java listing that shows how this can be achieved:

```
try {
      for (int i = 0; i < retries; i++) {
          try {
              return metrics.getFlowCount(strDN, strId, new
    HashMap());
          } catch (MetricsServiceException e) {
                  if (e.getErrorCode() == 100 //subsequent call
      error
                      || e.getErrorCode() == 200) { //too many
      clients
                      try {
                          Thread.sleep(retryPause);
                      }

      catch (Exception ex) {
                          // to nothing
                      }
                  } else {
                      throw e2;
                  }
```

```
                } else {
                    throw new RuntimeException(e);
                }
            } catch (Exception e) {
               throw e;
            }
         }
         throw new RuntimeException("Did not succeed making
    webservice call");
      } catch (Exception e) {
         throw e;
      }
   }
...
```

Metrics Web Service API

This section provides details about the methods available with the Metrics web service.

All of the methods throw MetricsServiceException and RemoteException. To improve readability, the throws clause has been omitted from the method signatures.

Team Manager Methods

This section provides reference information for each method available to team managers.

getClaimedFlowTimeCalendarDays

Syntax: Here's the method signature:

```
double getClaimedFlowTimeCalendarDays(String processId, String
processVersion, Date startCompletionTime, Date endCompletionTime, String
teamDN,Map filters)
```

getClaimedFlowTimeWorkingDays

Syntax: Here is the method signature:

```
MetricsResultset getClaimedFlowTimeWorkingDays(String processId, String
processVersion, Date startCompletionTime, Date endCompletionTime,String
teamDN, Map filters)
```

getToClaimedFlowTimeCalendarDays

Syntax: Here is the method signature:

```
 double getToClaimFlowTimeCalendarDays(String processId, String
processVersion, Date startCompletionTime, Date endCompletionTime, String
teamDN,Map filters)
```

getToClaimedFlowTimeWorkingDays

Syntax: Here is the method signature:

```
  MetricsResultset getToClaimFlowTimeWorkingDays(String processId, String
processVersion, Date startCompletionTime, Date endCompletionTime,String
teamDN, Map filters)
```

getClaimedInventory

Syntax: Here is the method signature:

```
 double getClaimedInventory(String processId, String processVersion, Date
startCompletionTime, Date endCompletionTime, String teamDN, Map filters)
```

getClaimedThroughputCalendarDays

Syntax: Here is the method signature:

```
  double getClaimedThroughputCalendarDays(String processId, String
processVersion, Date startCompletionTime, Date endCompletionTime, String
teamDN Map filters)
```

getClaimedThroughputWorkingDays

Syntax: Here is the method signature:

```
    MetricsResultset getClaimedThroughputWorkingDays(String processId,
String processVersion, Date startCompletionTime, Date endCompletionTime,
String teamDN, Map filters)
```

getTeamLongestRunning

Syntax: Here is the method signature:

```
MetricsResultset getTeamLongestRunning(String processId, String
processVersion, String teamDN, Map filters)
```

getTeamLongestClaimed

Syntax: Here is the method signature:

```
MetricsResultset getTeamLongestClaimed(String processId, String
processVersion, String teamDN, Map filters)
```

getTeamFlowHistory

Syntax: Here is the method signature:

```
MetricsResultset getTeamFlowHistory(List requestIds)
```

getTeamHistoryForInitiators

Syntax: Here is the method signature:

```
MetricsResultset getTeamHistoryForInitiators(String teamDN, Map filters)
```

getTeamHistoryForRecipients

Syntax: Here is the method signature:

```
MetricsResultset getTeamHistoryForRecipients(String teamDN, Map filters)
```

getTeamRunningTime

Syntax: Here is the method signature:

```
double getTeamRunningTime(String processId, String processVersion, String
teamDN, Map filters)
```

getTeamDecisionCount

Syntax: Here is the method signature:

```
int getTeamDecisionCount(String processId, String processVersion, String
teamDN, Map filters)
```

getTeamInitiatedCount

Syntax: Here is the method signature:

```
 int getTeamInitiatedCount(String processId, String processVersion, String
teamDN, Map filters)
```

getTeamRecipientCount

Syntax: Here is the method signature:

```
 int getTeamRecipientCount(String processId, String processVersion, String
teamDN, Map filters)
```

Provisioning Application Administrator Methods

This section provides reference information for each method available to the Provisioning Application Administrator.

getActivityFlowTimeCalendarDays

Syntax: Here is the method signature:

```
double getActivityFlowTimeCalendarDays(String processId, String
processVer, String activityId, Date startTime, Date completeTime, Map
filters)
```

getActivityFlowTimeWorkingDays

Syntax: Here is the method signature:

```
    MetricsResultset getActivityFlowTimeWorkingDays(String processId,
String processVer, String activityId, Date startTime, Date completeTime,
Map filters)
```

getActivityInventory

Syntax: Here is the method signature:

```
double getActivityInventory(String processId, String processVersion,
String activityId, Date startTime, Date completeTime, Map filters)
```

getActivityThroughputCalendarDays

Syntax: Here is the method signature:

```
double getActivityThroughputCalendarDays(String processId, String
processVersion, String activityId, Date startTime, Date completiontime, Map
filters)
```

getActivityThroughputWorkingDays

Syntax: Here is the method signature:

```
 MetricsResultset getActivityThroughputWorkingDays(String processId,
String processVersion, String activityId, Date startTime, Date
completiontime,    Map filters)
```

getInventory

Syntax: Here is the method signature:

```
double getInventory(String processId, String processVersion, Date
startTime, Date completeTime, Map filters)
```

getLongestClaimed

Syntax: Here is the method signature:

```
 MetricsResultset getLongestClaimed(String processId, String
processVersion, Map filters)
```

getLongestRunning

Syntax: Here is the method signature:

```
MetricsResultset getLongestRunning(String processId, String
processVersion, Map filters)
```

getFlowCount

Syntax: Here is the method signature:

```
int getFlowCount(String processId, String processVersion, Map filters)
```

getFlowHistory

Syntax: Here is the method signature:

```
MetricsResultset getFlowHistory(List requestIds)
```

getFlowHistoryForInitiators

Syntax: Here is the method signature:

```
    MetricsResultset getFlowHistoryForInitiators(List initiators, Map
filters)
```

getFlowHistoryForRecipients

Syntax: Here is the method signature:

```
MetricsResultset getFlowHistoryForRecipients(List recipients, Map filters)
```

getRunningTime

Syntax: Here is the method signature:

```
 double getRunningTime(String processId, String processVersion, Map
filters)
```

getThroughputCalendarDays

Syntax: Here is the method signature:

```
double getThroughputCalendarDays(String processId, String processVersion,
Date startTime, Date completiontime, Map filters)
```

getThroughputWorkingDays

Syntax: Here is the method signature:

```
MetricsResultset getActivityThroughputWorkingDays(String processId, String
processVersion, String activityId, Date startTime, Date completiontime,
Map filters)
```

Utility Methods

This section provides reference information for each utility method. Both team managers and administrators can call these methods.

getVersion

Syntax: Here is the method signature:

```
VersionVO getVersion()
```

getAllProvisioningFlows

Syntax: Here is the method signature:

```
MetricsResultset getAllProvisioningFlows()
```

getUserActivityOnlyFlow

Syntax: Here is the method signature:

```
BasicModelVO getUserActivityOnlyFlow(String processId, String processVer)
```

getTeams

Syntax: Here is the method signature:

```
MetricsResultset getTeams()
```

getTeamMembers

Syntax: Here is the method signature:

```
MetricsResultset getTeamMembers(String teamDN)
```

Metrics Web Service Examples

This section provides examples that show how to use the Metrics Web Service to gather workflow metrics. The examples assume that you have obtained a stub, as shown in "Obtaining the Remote Interface" on page 338, and potentially wrapped it in an object that handles the potential error conditions, as described in "Metrics Configuration Settings" on page 339.

General Examples

This example uses the KEY_APPROVAL_STATUS filter to compare the decision outcomes for a provisioning request type. This could be used to generate a pie chart for example.

```
FilterConstants constants=new FilterConstants();
Map<MetricsFilter, Object> map = new HashMap<MetricsFilter, Object>();
map.put(MetricsFilter.KEY_APPROVAL_STATUS,constants.getApprovalStatusAppro
ved());
double accepted=stubWrapper.getFlowCount(processId,processVersion,map);
map.put(MetricsFilter.KEY_APPROVAL_STATUS,constants.getApprovalStatusDenie
d());
double denied=stubWrapper.getFlowCount(processId,processVersion,map);
map.put(MetricsFilter.KEY_APPROVAL_STATUS,constants.getApprovalStatusError
());
double error=stubWrapper.getFlowCount(processId,processVersion,map);
map.put(MetricsFilter.KEY_APPROVAL_STATUS,constants.getApprovalStatusRetra
ct());
double retracted=stubWrapper.getFlowCount(processId,processVersion,map);
map.put(MetricsFilter.KEY_APPROVAL_STATUS,
constants.getApprovalStatusRefused());
double refused = stubWrapper.getFlowCount(processId,
processVersion, map);
```

Additional filters may be specified by adding appropriate entries to the filter map. The following examples illustrate how you might add various types of filters.

Adding a start date filter

To add a start date filter (01/01/2006 < date < 02/01/2006):

```
Calendar startDate=Calendar.getInstance();
startDate.set(2006,0,1);
Calendar endDate=Calendar.getInstance();
endDate.set(2006,1,1);
map.put(MetricsFilter.KEY_L_START_TIME,startDate);
map.put(MetricsFilter.KEY_S_START_TIME,endDate)
```

Adding a completion date filter

To add a completion date filter (02/01/2005 < date <03/01/2005)

```
Calendar startDate=Calendar.getInstance();
startDate.set(2006,0,1);
Calendar endDate=Calendar.getInstance();
endDate.set(2006,1,1);
map.put(MetricsFilter.KEY_L_COMPLETION_TIME,startDate);
map.put(MetricsFilter.KEY_S_COMPLETION_TIME,endDate)
```

Narrowing requests to a specific initiator

To narrow down counted requests to a specific initiator

```
map.put(MetricsFilter.KEY_INITIATOR,"cn=admin,ou=idmsample,o=netiq");
```

Narrowing requests to a specific recipient

To narrow down counted requests to a specific recipient

```
map.put(MetricsFilter.KEY_RECIPIENT,"cn=admin,ou=idmsample,o=netiq");
```

Other Examples

The following examples illustrate the use of various methods for retrieving workflow counts.

Retrieving decision counts for a team

This example describes how to retrieve the various decision outcomes of a team. The team's DN is required and can be obtained by using the getTeams() method:

```
FilterConstants constants=new FilterConstants();
Map<MetricsFilter, Object> map = new HashMap<MetricsFilter, Object>();
map.put(MetricsFilter.KEY_ACTIVITY_END,
constants.getActivityApproved());
double accepted = stubWrapper.getTeamDecisionCount(processId,
processVersion, teamDN, map);
map.put(MetricsFilter.KEY_ACTIVITY_END,
constants.getActivityDenied());
double denied = stubWrapper.getTeamDecisionCount(processId,
processVersion, teamDN, map);
map.put(MetricsFilter.KEY_ACTIVITY_END,
         constants.getActivityReassigned());
double reassigned = stubWrapper.getTeamDecisionCount(processId,
         processVersion, teamDN, map);
map.put(MetricsFilter.KEY_ACTIVITY_END,
         constants.getActivityRefused());
double refused = stubWrapper.getTeamDecisionCount(processId,
         processVersion, teamDN, map);
```

Retrieving decision counts for requests where team members are recipients

This example describes how to retrieve the various decisions outcomes for requests for which members of the team act as recipients

```
FilterConstants constants = new FilterConstants();
Map<MetricsFilter, Object> map = new HashMap<MetricsFilter, Object>();
map.put(MetricsFilter.KEY_APPROVAL_STATUS,
constants.getActivityApproved());
double accepted = stubWrapper.getTeamRecipientCount(processId,
processVersion, teamDN, map);
map.put(MetricsFilter.KEY_APPROVAL_STATUS,
constants.getApprovalStatusDenied());
double denied = stubWrapper.getTeamRecipientCount(processId,
processVersion, teamDN, map);
map.put(MetricsFilter.KEY_APPROVAL_STATUS,
constants.getApprovalStatusError());
double error = stubWrapper.getTeamRecipientCount(processId,
processVersion, teamDN, map);
map.put(MetricsFilter.KEY_APPROVAL_STATUS,
constants.getApprovalStatusError());
double retracted = stubWrapper.getTeamRecipientCount(processId,
processVersion, teamDN, map);
map.put(MetricsFilter.KEY_APPROVAL_STATUS,
constants.getApprovalStatusRefused());
double refused = stubWrapper.getTeamRecipientCount(processId,
processVersion, teamDN, map);
```

Retrieving requests that have been claimed but not acted on

This example describes how to retrieve the requests started after 03/01/2006 that have been claimed but not acted upon.

```
Map<MetricsFilter, Object> map = new HashMap<MetricsFilter, Object>();
Calendar startDate=Calendar.getInstance();
startDate.set(2006,2,1);
map.put(MetricsFilter.KEY_L_START_TIME,startDate);
MetricsResultset rset = stubWrapper.getLongestClaimed(processId,
processVersion, map);
```

Retrieving the longest running requests started by a particular user

This example describes how to retrieve the longest running requests that have been started by initiator "cn=admin,ou=idmsample,o=netiq";

```
Map<MetricsFilter, Object> map = new HashMap<MetricsFilter, Object>();
map.put(MetricsFilter.KEY_INITIATOR,""cn=admin,ou=idmsample,o=netiq");
MetricsResultset rset = stubWrapper.getLongestRunning(processId,
processVersion, map);
```

Retrieving activity inventory

This example describes the average inventory for users handling decision with activity id "managerApproval" between 01/01/2006 and 02/01/2006

```
Map<MetricsFilter, Object> map = new HashMap<MetricsFilter, Object>();
Calendar startDate=Calendar.getInstance();
startDate.set(2006,0,1);
Calendar endDate=Calendar.getInstance();
endDate.set(2006,1,1);
MetricsResultset rset = stubWrapper.getActivityInventory(processId,
processVersion,"managerApproval", startDate, endDate, map );
```

Retrieving the Claimed Throughput and Inventory for a Team

This example describes the team's throughput and inventory over the time interval between 01/01/2006 and 02/01/2006

```
Map<MetricsFilter, Object> map = new HashMap<MetricsFilter, Object>();
Calendar startDate=Calendar.getInstance();
startDate.set(2006,0,1);
Calendar endDate=Calendar.getInstance();
endDate.set(2006,1,1);
double throughput =
stubWrapper.getClaimedThroughputCalendarDays(processId, processVersion,
startDate, endDate,teamDN, map);
double inventory = stubWrapper.getClaimedInventory(processId,
processVersion, startDate, endDate, teamDN, map)
```

27 Notification Web Service

This section describes the Notification Web Service, which allows SOAP clients to use the email notification facility.

About the Notification Web Service

The Identity Manager User Application includes an email notification facility that lets you send email messages to notify users of changes in the state of the provisioning system, as well as tasks that they need to perform. To support access by third-party software applications, the notification facility includes a Web service endpoint. The endpoint lets you send an email message to one or more users. When you send an email, you include parameters that specify the target email address, the email template to use, and the replacement values for tokens in the email template.

This Appendix describes the programming interface for the Notification Web Service.

Accessing the Test Page

You can access the Notification Web Service endpoint using a URL similar to the following:

`http://server:port/warcontext/notification/service?test`

For example, if your server is named “myserver”, your User Application is listening on port 8080, and your User Application war file is named “IDMPROV”, the URL would be:

`http://myserver:8080/IDMPROV/notification/service?test`

You can also access the SOAP endpoint by going to the **Administration** within the User Application. To do this, you need to select the **Application Configuration** tab, then select **Web Services** from the left-navigation menu. After selecting **Web Services**, pick the Web Service endpoint you want from the list.

WARNING: The test page is disabled by default. Since some of the methods allow data to be updated, the test page presents a potential security vulnerability and should not be allowed in a production environment. For details on enabling the test page, see the instructions provided for the Role Service in “Enabling the Test Page” on page 488.

Accessing the WSDL

You can access the WSDL for the Notification Web Service using a URL similar to the following:

`http://server:port/warcontext/notification/service?wsdl`

For example, if your server is named “myserver”, your User Application is listening on port 8080, and your User Application war file is named “IDMPROV”, the URL would be:

`http://myserver:8080/IDMPROV/notification/service?wsdl`

Generating the Stub Classes

Before using the Web Service, you need to use the WSSDK tool or another SOAP tool kit to generate the stub classes. To allow your code to find the stub classes, you also need to add the JAR that contains the stub classes to your classpath.

If you want to use the NetIQ WSSDK tool, you can generate the client stubs by extracting the WSDL and running the wsdl2java utility. For example, you could run this command to generate the stubs in a package called com.novell.ws.client.notification:

```
"C:\Program Files\Java\jdk1.6.0_31\bin\java" -cp "../lib/wssdk.jar;../lib/
jaxrpc-api.jar";"../lib/mail.jar";"../lib/activation.jar";"c:\Program
Files\Java\jdk1.6.0_31\lib\tools.jar";
com.novell.soa.ws.impl.tools.wsdl2java.Main -verbose -ds gensrc -d C:\ -
noskel -notie -genclient -keep -package com.novell.ws.client.notification -
javadoc notification.wsdl
```

You can change the wsdl2java parameters to suit your requirements.

Notification Web Service API

This section provides details about the methods available with the Notification Web service. This API presumes you're using Java code generated by the WSSDK toolkit. The API will be different if you're using another Web Service toolkit.

All of the methods throw RemoteException. To improve readability, the throws clause has been omitted from the method signatures.

iRemoteNotification

This section provides reference information for each method associated with the iRemoteNotification interface.

getVersion

Returns the version number of the notification facility you're running.

Syntax: Here is the method signature:

```
VersionVO getVersion()
```

sendNotification

Sends an email notification.

Syntax: Here is the method signature:

```
void sendNotification(NotificationMap arg0)
```

BuiltInTokens

This section provides reference information for each method associated with the BuiltInTokens class.

BuiltInTokens constructor

The BuiltInTokens class has a single constructor.

Syntax: Here is the constructor for the BuiltInTokens class:

```
BuiltInTokens()
```

getTO

Returns the fixed string TO, which can be used as a key to identify the value for the TO system token.

Syntax: Here is the method signature:

```
public java.lang.String getTO()
```

getCC

Returns the fixed string CC, which can be used as a key to identify the value for the CC system token.

Syntax: Here is the method signature:

```
public java.lang.String getCC()
```

getBCC

Returns the fixed string BCC, which can be used as a key to identify the value for the BCC system token.

Syntax: Here is the method signature:

```
public java.lang.String getBCC()
```

getTO_DN

Returns the fixed string TO_DN, which can be used as a key to identify the value for the TO_DN system token.

Syntax Here is the method signature:

```
public java.lang.String getTO_DN()
```

getCC_DN

Returns the fixed string CC_DN, which can be used as a key to identify the value for the CC_DN system token.

Syntax: Here is the method signature:

```
public java.lang.String getCC_DN()
```

getBCC_DN

Returns the fixed string BCC_DN, which can be used as a key to identify the value for the BCC_DN system token.

Syntax: Here is the method signature:

```
public java.lang.String getBCC_DN()
```

getREPLYTO

Returns the fixed string REPLYTO, which can be used as a key to identify the value for the REPLYTO system token.

Syntax: Here is the method signature:

```
public java.lang.String getREPLYTO()
```

getREPLYTO_DN

Returns the fixed string REPLYTO_DN, which can be used as a key to identify the value for the REPLYTO_DN system token.

Syntax: Here is the method signature:

```
public java.lang.String getREPLYTO_DN()
```

getLOCALE

Returns the fixed string LOCALE, which can be used as a key to identify the value for the LOCALE system token.

Syntax: Here is the method signature:

```
public java.lang.String getLOCALE()
```

getNOTIFICATION_TEMPLATE_DN

Returns the fixed string NOTIFICATION_TEMPLATE, which can be used as a key to identify the value for the NOTIFICATION_TEMPLATE system token.

Syntax: Here is the method signature:

```
public java.lang.String getNOTIFICATION_TEMPLATE_DN()
```

Entry

The Entry class represents an entry in an EntryArray object. It is used to specify a token in an email template.

This section provides reference information for each method associated with the Entry class.

Entry constructors

The Entry class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
Entry()
```

Syntax 2: Here is the syntax for a constructor that takes two parameters, the key value and an array of values:

```
Entry(java.lang.String KeyVal, StringArray ValuesVal)
```

getKey

Returns the key defined for the Entry object. The key identifies the token.

Syntax: Here is the method signature:

```
java.lang.String getKey()
```

setKey

Sets the key for the Entry object. The key identifies the token. If the object represents a built-in token, you can use the BuiltInTokens class to set the key. Otherwise, you can pass a string to the setKey method that specifies the key.

Syntax: Here is the method signature:

```
void setKey(java.lang.String KeyVal)
```

getValues

Returns a StringArray object representing the values for the Entry object.

Syntax: Here is the method signature:

```
StringArray getValues()
```

setValues

Sets the values for the Entry object.

Syntax: Here is the method signature:

```
void setValues(StringArray ValuesVal)
```

EntryArray

The EntryArray class is a container for an array of Entry objects. It is contained by the NotificationMap object.

This section provides reference information on the methods associated with the EntryArray class.

EntryArray constructors

The EntryArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
EntryArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of Entry objects as a parameter:

```
EntryArray(Entry[] EntryVal)
```

getEntry

Returns the Entry object contained within this EntryArray object.

Syntax: Here is the method signature:

```
Entry[] getEntry()
```

setEntry

Sets the Entry object for this EntryArray object.

Syntax: Here is the method signature:

```
void setEntry(Entry[] EntryVal)
```

NotificationMap

The NotificationMap object is a map that contains an EntryArray object. It is passed to the sendNotification method on the stub.

This section provides reference information for the methods associated with the NotificationMap class.

NotificationMap constructors

The NotificationMap class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
NotificationMap()
```

Syntax 2: Here is the syntax for a constructor that takes an EntryArray object as a parameter:

```
NotificationMap(EntryArray EntriesVal)
```

getEntries

Returns the EntryArray object contained by this NotificationMap object.

Syntax: Here is the method signature:

```
EntryArray getEntries()
```

setEntries

Sets the EntryArray object for this NotificationMap object.

Syntax: Here is the method signature:

```
void setEntries(EntryArray EntriesVal)
```

NotificationService

This section provides reference information for the NotificationService interface.

getIRemoteNotificationPort

Gets the stub for the remote service. The stub is a port of type IRemoteNotification.

Syntax: Here is the method signature:

```
IRemoteNotification getIRemoteNotificationPort() throws
javax.xml.rpc.ServiceException
```

StringArray

This section provides reference information for the StringArray class.

StringArray constructors

The StringArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
StringArray()
```

Syntax 2: Here is the syntax for a constructor that takes a String array as a parameter:

```
StringArray(java.lang.String[] StringVal)
```

getString

Returns the array of strings defined for this StringArray object.

Syntax: Here is the method signature:

```
java.lang.String[] getString()
```

setString

Sets the array of strings for this StringArray object. This method is called by the second constructor, which takes a String array as a parameter.

Syntax: Here is the method signature:

```
void setString(java.lang.String[] StringVal)
```

VersionVO

This section provides reference information on the VersionVO class.

getValue

Returns the version number of the service.

Syntax: Here is the method signature:

```
java.lang.String getValue()
```

Notification Example

The following code example shows how one might use the Notification service to send an email message using a pre-defined system template. To get a reference to the SOAP endpoint for the Notification service, a call is made to the getNotificationStub() method. After acquiring the stub interface, the code sets the email notification template as well as values for the built-in tokens in the template. In addition, the code specifies values for the requestTitle and initiatorFullName tokens. For each token, the code creates an Entry object. Once all of the entries have been created, it packages the entry array into a map of type NotificationMap, which is then passed to the sendNotification method on the stub.

```
import java.util.Properties;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.xml.rpc.Stub;
import java.rmi.RemoteException;
//
// Notification imports
import com.novell.ws.client.notification.IRemoteNotification;
import com.novell.ws.client.notification.BuiltInTokens;
import com.novell.ws.client.notification.Entry;
import com.novell.ws.client.notification.EntryArray;
import com.novell.ws.client.notification.StringArray;
import com.novell.ws.client.notification.NotificationMap;
import com.novell.ws.client.notification.IRemoteNotification;
import com.novell.ws.client.notification.NotificationService;


public class NotificationTest
{
  private static final int LOCALHOST    = 0;  // localhost
  private static final int TESTSERVER      = 1;  // testserver
  private static final int SELECTED_URL     = TESTSERVER;

      private String [] SERVER_URLS = {
    "http://localhost:8080/IDMProv/notification/service",
    "http://testserver:8080/IDMProv/notification/service"
  };
  private String url = SERVER_URLS[SELECTED_URL];
  private String username = "cn=admin,ou=idmsample,o=netiq";
```

```
  private String password = "test";

  public void emailNotificationTestCase()
  throws Exception
  {
        System.out.println("\nCalling emailNotificationTestCase() test
case");

        try
        {
    String targetEmailAddress = "jsmith@somewhere.com";
            //
            // Get the notification stub
            IRemoteNotification notificationStub =
getNotificationStub(url, username, password);

            BuiltInTokens builtInTokens = new BuiltInTokens();
            //
            // Set the To: entry
            Entry to = new Entry();
            to.setKey(builtInTokens.getTO());
            StringArray arr = new StringArray(new
String[]{targetEmailAddress} );
            to.setValues(arr);
            //
            // Set which email template to use : list in iManager
(Workflow Admin->Email Templates)
            Entry notificationTemplate = new Entry();

notificationTemplate.setKey(builtInTokens.getNOTIFICATION_TEMPLATE_DN());
            //
            // Use one of the email templates specifying DN
            String EMAIL_TEMPLATE_NAME = "Provisioning Notification";
            String templateDN = "cn=" + EMAIL_TEMPLATE_NAME +
",cn=Default Notification Collection,cn=Security";
            arr = new StringArray(new String[]{templateDN} );
            notificationTemplate.setValues(arr);
            //
            // Substitute key values defined in email templates
            Entry token1 = new Entry();
            token1.setKey("requestTitle"); // key is %requestTitle%
            arr = new StringArray(new String[]{"Sample Email using
Notification Web Service"} );
            token1.setValues(arr);
            Entry token2 = new Entry();
            token2.setKey("initiatorFullName");
            arr = new StringArray(new String[]{username} );
            token2.setValues(arr);
            //
            // Setup the notification map
            NotificationMap map = new NotificationMap();
            Entry[] entries = new
Entry[]{to,notificationTemplate,token1,token2};
            EntryArray entryArray = new EntryArray();
            entryArray.setEntry(entries);
```

```
            map.setEntries(entryArray);
            //
            // Make the notification endpoint call
            notificationStub.sendNotification(map);
        }
        catch(RemoteException error)
        {
            System.out.println(error.getMessage() );
            throw new Exception(error.getMessage() );
        }
     }


   /**
     * Method to obtain the remote interface to the Notification
endpoint
     * @param _url
     * @param _username
     * @param _password
     * @return IRemoteNotification interface
     * @throws Exception
     */
    private IRemoteNotification getNotificationStub(String _url,
String _username, String _password)
    throws Exception
    {
        Properties properties = new Properties();
        properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jnp.interfaces.NamingContextFactory");

        String lookup =
"xmlrpc:soap:com.novell.ws.client.notification.NotificationService";

        InitialContext ctx = new InitialContext();
        NotificationService svc = (NotificationService)
ctx.lookup(lookup);

        Stub stub = (Stub)svc.getIRemoteNotificationPort();

        stub._setProperty(Stub.USERNAME_PROPERTY, _username);
        stub._setProperty(Stub.PASSWORD_PROPERTY, _password);
        stub._setProperty(Stub.SESSION_MAINTAIN_PROPERTY,
Boolean.TRUE);
        stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, _url);

        return (IRemoteNotification) stub;
    }

}
```

28 Directory Abstraction Layer (VDX) Web Service

This section describes the VDX Web Service, which allows SOAP clients to access the directory abstraction layer.

About the Directory Abstraction Layer (VDX) Web Service

The directory abstraction layer provides a logical view of the Identity Vault data. To support access by third-party software applications, the directory abstraction layer includes a Web service endpoint called the VDX Web Service. This endpoint lets you access the attributes associated with entities defined in the directory abstraction layer. It also lets you perform ad hoc searches for entities and execute predefined searches called global queries. You can think of global queries as stored procedures for LDAP.

This Appendix describes the programming interface for the VDX Web Service.

Accessing the Test Page

You can access the VDX Web Service endpoint using a URL similar to the following:

`http://server:port/warcontext/vdx/service?test`

For example, if your server is named “myserver”, your User Application is listening on port 8080, and your User Application war file is named “IDMPROV”, the URL would be:

`http://myserver:8080/IDMPROV/vdx/service?test`

You can also access the SOAP endpoint by going to the **Administration** within the User Application. To do this, you need to select the **Application Configuration** tab, then select **Web Services** from the left-navigation menu. After selecting **Web Services**, pick the Web Service endpoint you want from the list.

WARNING: The test page is disabled by default. Since some of the methods allow data to be updated, the test page presents a potential security vulnerability and should not be allowed in a production environment. For details on enabling the test page, see the instructions provided for the Role Service in “Accessing the Test Page” on page 351.

Accessing the WSDL

You can access the WSDL for the VDX Web Service using a URL similar to the following:

`http://server:port/warcontext/vdx/service?wsdl`

For example, if your server is named “myserver”, your User Application is listening on port 8080, and your User Application war file is named “IDMPROV”, the URL would be:

```
http://myserver:8080/IDMPROV/vdx/service?wsdl
```

Generating the Stub Classes

Before using the Web Service, you need to use the WSSDK tool or another SOAP tool kit to generate the stub classes. To allow your code to find the stub classes, you also need to add the JAR that contains the stub classes to your classpath.

If you want to use the NetIQ WSSDK tool, you can generate the client stubs by extracting the WSDL and running the wsdl2java utility. For example, you could run this command to generate the stubs in a package called com.novell.ws.client.vdx:

```
"C:\Program Files\Java\jdk1.6.0_31\bin\java" -cp "../lib/wssdk.jar;../lib/
jaxrpc-api.jar";"../lib/mail.jar";"../lib/activation.jar";"c:\Program
Files\Java\jdk1.8.0_162\lib\tools.jar";
com.novell.soa.ws.impl.tools.wsdl2java.Main -verbose -ds gensrc -d C:\ -
noskel -notie -genclient -keep -package com.novell.ws.client.vdx -javadoc
vdx.wsdl
```

You can change the wsdl2java parameters to suit your requirements.

Removing Administrator Credential Restrictions

The VDX Web Service supports two levels of security, one that restricts access to Provisioning Administrators and another that restricts access to the authenticated user. The default setting restricts access to all operations to the Provisioning Administrator.

You can modify the settings for security configuration in the `ism-configuration.properties` file, located by default in the `/netiq/idm/apps/tomcat/conf` directory. . Each property can be set to true or false. A value of true locks down the operation, whereas a value of false opens up the operation.

You can open up the VDX Web Service to authenticated users by setting the VirtualDataService/soap property to false. To open up a particular operation to authenticated users, you need to set the property for that operation (VirtualDataService/soap/*operation*) to false as well. If you set all of the properties to false, you can open up all operations to authenticated users. The *operation* names are the same as the names of the methods supported by the service.

Example To ensure that the security configuration opens up all operations within the VDX Web Service, the `ism-configuration.properties` file must have the following setting:

```
VirtualDataService/soap = false
```

To restrict globalQuery, change the `VirtualDataService/soap/globalQuery` to `true` in the `ism-configuration.properties` file.

Even though the service does not require the Administrator credentials since you set the VirtualDataService/soap property to false, the globalQuery operation will still require the Administrator credentials since you set a property for the operation to true.

VDX Web Service API

This section provides details about the methods available with the VDX Web service. This API presumes you're using Java code generated by the WSSDK toolkit. The API will be different if you're using another Web Service toolkit.

All of the methods throw VdxServiceException. To improve readability, the throws clause has been omitted from the method signatures.

IRemoteVdx

This section provides reference information for each method associated with the IRemoteVdx interface.

getVersion

Returns the version number of the VDX service you're running.

Syntax: Here is the method signature:

```
VersionVO getVersion() throws java.rmi.RemoteException;
```

globalQuery

Allows you to execute predefined searches called global queries. Global queries are saved searches for LDAP. They provide some of the capabilities of stored procedures.

To define a global query, you need to use the directory abstraction layer editor. For details, see the chapter on the directory abstraction layer editor in the *Identity Manager User Application: Design Guide*.

Syntax: Here is the method signature:

```
java.lang.String[] globalQuery(java.lang.String queryDN, StringMap
queryParameterValues) throws VdxServiceException,
java.rmi.RemoteException;
```

query

Allows you to perform ad hoc queries by specifying an entity, a set of attributes, and a query expression that filters the data returned.

Syntax: Here is the method signature:

```
EntityAttributeMap query(java.lang.String entityDefinition,
java.lang.String[] attributeKeys, java.lang.String queryFilter) throws
VdxServiceException, java.rmi.RemoteException;
```

Query Grammar

The queryFilter parameter of the query() method lets you pass in search criteria expressions that filter the data returned. This section describes the grammar for these expressions.

Query syntax 1: The simplest form of a query is the following:

RelationalExpression1

Query syntax 2: A query can also combine relational expressions with a logical operator:

RelationalExpession1 logicalOperator RelationalExpression2

Query syntax 3: Alternatively, a query can use parentheses to set off the expressions:

(RelationalExpession1) logicalOperator (RelationalExpression2)

Query syntax 4: A query can also use parentheses to set off sub queries:

RelationalExpession1 logicalOperator (RelationalExpression2 logicalOperator1 RelationalExpression3)

Relational expressions must be separated by a logical operator which must remain the same. In other words, the following query is valid:

```
expression1 AND expression2 AND expression3
```

However, this query is not valid:

```
expression1 AND expression2 OR expression3
```

You can use parentheses to create a condition group, as in the following example:

```
expression1 AND (expression2 OR expression3)
```

Grammar for Relational Expressions

Relational expression syntax: A relational expression must conform to this syntax:

attribute relationalOperator value

Grammar for Operators and Values

Relational operators: The relational operator must be one of the following:

```
> , < , >= , <= , = , != , !< , !> , !<= , !>= , STARTWITH, !STARTWITH, IN
, !IN , PRESENT, !PRESENT
```

Logical operators: The logical operator must be one of the following:

```
AND, OR
```

Value: The value side of an expression must be one of the following:

```
'foo',"foo", 1-9, true, false
```

The PRESENT and !PRESENT relational operators require no value.

getAttribute

Returns a single Attribute object that can be used to retrieve and examine data for an attribute in the directory abstraction layer.

Syntax: Here is the method signature:

```
Attribute getAttribute(java.lang.String objectDN, java.lang.String
entityDefinition, java.lang.String attributeKey) throws
VdxServiceException, java.rmi.RemoteException;
```

getAttributes

Returns an array of Attribute objects that can be used to retrieve and examine data in the directory abstraction layer.

Syntax: Here is the method signature:

```
Attribute[] getAttributes(java.lang.String objectDN, java.lang.String
entityDefinition, java.lang.String[] attributeKeys) throws
VdxServiceException, java.rmi.RemoteException;
```

Attribute

The Attribute class represents an attribute in the directory abstraction layer.

This section provides reference information for the Attribute class.

Attribute constructors

The Attribute class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no arguments:

```
Attribute()
```

Syntax 2: Here is the syntax for a constructor that takes arrays of all the supported data types as arguments:

```
Attribute(ByteArrayArray BinariesVal, BooleanArray BooleansVal, DateArray
DatesVal, IntegerArray IntegersVal, StringArray StringsVal, AttributeType
TypeVal)
```

getBinaries

Returns the ByteArrayArray object for the attribute.

Syntax: Here is the method signature:

```
ByteArrayArray getBinaries()
```

setBinaries

Sets the ByteArrayArray object for the attribute.

Syntax: Here is the method signature:

```
void setBinaries(ByteArrayArray BinariesVal)
```

getBooleans

Returns the BooleanArray object for the attribute.

Syntax: Here is the method signature:

```
BooleanArray getBooleans()
```

setBooleans

Sets the BooleanArray object for the attribute.

Syntax: Here is the method signature:

```
void setBooleans(BooleanArray BooleansVal)
```

getDates

Returns the DateArray object for the attribute.

Syntax: Here is the method signature:

```
DateArray getDates()
```

setDates

Sets the DateArray object for the attribute.

Syntax: Here is the method signature:

```
void setDates(DateArray DatesVal)
```

getIntegers

Returns the IntegerArray object for the attribute.

Syntax: Here is the method signature:

```
IntegerArray getIntegers()
```

setIntegers

Sets the IntegerArray object for the attribute.

Syntax: Here is the method signature:

```
void setIntegers(IntegerArray IntegersVal)
```

getStrings

Returns the StringArray object for the attribute.

Syntax: Here is the method signature:

```
StringArray getStrings()
```

setStrings

Set the StringArray object for the attribute.

Syntax: Here is the method signature:

```
void setStrings(StringArray StringsVal)
```

getType

Returns the AttributeType object for the attribute.

Syntax: Here is the method signature:

```
AttributeType getType()
```

setType

Sets the AttributeType object for the attribute.

Syntax: Here is the method signature:

```
void setType(AttributeType TypeVal)
```

AttributeArray

This section provides reference information on the AttributeArray class.

AttributeArray constructors

The AttributeArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
AttributeArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of Attribute objects as a parameter:

```
AttributeArray(Attribute[] AttributeVal)
```

getAttribute

Returns an array of Attribute objects.

Syntax: Here is the method signature:

```
Attribute[] getAttribute()
```

setAttribute

Sets the array of Attribute objects associated with the AttributeArray class.

Syntax: Here is the method signature:

```
void setAttribute(Attribute[] AttributeVal)
```

AttributeType

This section provides reference information on the AttributeType class.

AttributeType constructors

The AttributeType class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
protected AttributeType(java.lang.String value)
```

getValue

Returns a String that indicates the attribute type.

Syntax: Here is the method signature:

```
java.lang.String getValue()
```

BooleanArray

This section provides reference information for the BooleanArray class.

BooleanArray constructors

The BooleanArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
BooleanArray()
```

Syntax 2: Here is the syntax for a constructor that takes a boolean value as a parameter:

```
BooleanArray(boolean[] BooleanVal)
```

getBoolean

Returns an array of boolean values for an attribute.

Syntax: Here is the method signature:

```
boolean[] getBoolean()
```

setBoolean

Sets an array of boolean values for an attribute.

Syntax: Here is the method signature:

```
void setBoolean(boolean[] BooleanVal)
```

ByteArrayArray

This section provides reference information on the ByteArrayArray class.

ByteArrayArray constructors

The ByteArrayArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
ByteArrayArray()
```

Syntax 2: Here is the syntax for a constructor that takes a Base 64 binary value as a parameter:

```
ByteArrayArray(byte[][] Base64BinaryVal)
```

getBase64Binary

Returns a two-dimensional array of bytes for an attribute.

Syntax: Here is the method signature:

```
byte[][] getBase64Binary()
```

setBase64Binary

Sets a two-dimensional array of bytes for an attribute.

Syntax: Here is the method signature:

```
void setBase64Binary(byte[][] Base64BinaryVal)
```

DateArray

This section provides reference information for the DateArray class.

DateArray constructors

The DateArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
DateArray()
```

Syntax 2: Here is the syntax for a constructor that takes a Calendar array as a parameter:

```
DateArray(java.util.Calendar[] DatetimeVal)
```

getDatetime

Returns an array of Calendar objects for an attribute.

Syntax: Here is the method signature:

```
java.util.Calendar[] getDatetime()
```

setDatetime

Sets an array of Calendar objects for an attribute.

Syntax: Here is the method signature:

```
void setDatetime(java.util.Calendar[] DatetimeVal)
```

EntryAttributeMap

The EntryAttributeMap class is a container for an EntryArray object. It is returned by the query method on the stub.

This section provides reference information on the methods associated with the EntryAttributeMap class.

EntryAttributeMap constructors

The EntryAttributeMap class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
EntryAttributeMap()
```

Syntax 2: Here is the syntax for a constructor that takes an EntryArray object as a parameter:

```
EntityAttributeMap(EntryArray EntriesVal)
```

getEntries

Returns the EntryArray object contained within this EntryAttributeMap object.

Syntax: Here is the method signature:

```
EntryArray getEntries()
```

setEntries

Sets the EntryArray object for this EntryAttributeMap object.

Syntax: Here is the method signature:

```
void setEntry(EntryArray EntriesVal)
```

Entry

The Entry class represents an entry in an EntryArray object.

This section provides reference information for each method associated with the Entry class.

Entry constructors

The Entry class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
Entry()
```

Syntax 2: Here is the syntax for a constructor that takes two parameters, the key value and an array of attribute values:

```
Entry(java.lang.String KeyVal, AttributeArray ValuesVal)
```

getKey

Returns the key defined for the Entry object. The key identifies the attribute.

Syntax: Here is the method signature:

```
java.lang.String getKey()
```

setKey

Sets the key for the Entry object. The key identifies the attribute.

Syntax: Here is the method signature:

```
void setKey(java.lang.String KeyVal)
```

getValues

Returns a AttributeArray object representing the values for the Entry object.

Syntax: Here is the method signature:

```
AttributeArray getValues()
```

setValues

Sets the values for the Entry object.

Syntax: Here is the method signature:

```
void setValues(AttributeArray ValuesVal)
```

EntryArray

The EntryArray class is a container for an array of Entry objects. It is contained by the EntryAttributeMap object.

This section provides reference information on the methods associated with the EntryArray class.

EntryArray constructors

The EntryArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
EntryArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of Entry objects as a parameter:

```
EntryArray(Entry[] EntryVal)
```

getEntry

Returns the Entry object contained within this EntryArray object.

Syntax: Here is the method signature:

```
Entry[] getEntry()
```

setEntry

Sets the Entry object for this EntryArray object.

Syntax: Here is the method signature:

```
void setEntry(Entry[] EntryVal)
```

IntegerArray

This section provides reference information for the IntegerArray class.

IntegerArray constructors

The IntegerArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
IntegerArray()
```

Syntax 2: Here is the syntax for a constructor that takes an int array as a parameter:

```
IntegerArray(int[] IntVal)
```

getInt

Returns an array of integers for an attribute.

Syntax: Here is the method signature:

```
int[] getInt()
```

setInt

Sets an array of integers for an attribute.

Syntax: Here is the method signature:

```
void setInt(int[] IntVal)
```

StringArray

The StringArray class is a container for an array of String objects. When you call the query() and getAttributes() methods, you pass in a StringArray object to specify which attributes you want to retrieve values for.

This section provides reference information for the StringArray class.

StringArray constructors

The StringArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
StringArray()
```

Syntax 2: Here is the syntax for a constructor that takes an String array as a parameter:

```
StringArray(java.lang.String[] StringVal)
```

getString

Returns the array of String objects associated with the StringArray object.

Syntax: Here is the method signature:

```
java.lang.String[] getString()
```

setString

Sets the array of String objects associated with the StringArray object.

Syntax: Here is the method signature:

```
void setString(java.lang.String[] StringVal)
```

StringEntry

The StringEntry class is contained by the the StringEntryArray class.

This section provides reference information for the StringEntry class.

StringEntry constructors

The StringEntry class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
StringEntry()
```

Syntax 2: Here is the syntax for a constructor that takes a key and a String value as parameters:

```
StringEntry(java.lang.String KeyVal, java.lang.String ValuesVal)
```

getKey

Returns the key defined for the StringEntry object.

Syntax: Here is the method signature:

```
java.lang.String getKey()
```

setKey

Sets the key for the StringEntry object.

Syntax: Here is the method signature:

```
void setKey(java.lang.String KeyVal)
```

StringEntryArray

The StringEntryArray class is a container for an array of StringEntry objects. It is contained by the StringMap object.

This section provides reference information for the StringEntryArray class.

StringEntryArray constructors

The StringEntryArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
StringEntryArray()
```

Syntax 2: Here is the syntax for a constructor that takes a StringEntry array as a parameter:

```
StringEntryArray(StringEntry[] StringentryVal)
```

getStringentry

Returns the key for the StringEntryArray object.

Syntax: Here is the method signature:

```
StringEntry[] getStringentry()
```

setStringentry

Sets the key for the StringEntryArray object.

Syntax: Here is the method signature:

```
void setStringentry(StringEntry[] StringentryVal)
```

StringMap

The StringMap is a container for a StringEntryArray object.

This section provides reference information on the StringMap class.

StringMap constructors

The StringMap class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
StringMap()
```

Syntax 2: Here is the syntax for a constructor that takes a StringEntryArray as a parameter:

```
StringMap(StringEntryArray EntriesVal)
```

getEntries

Returns the StringEntryArray object contained by this StringMap object.

Syntax: Here is the method signature:

```
StringEntryArray getEntries()
```

setEntries

Sets the StringEntryArray object for this StringMap object.

Syntax: Here is the method signature:

```
void setEntries(StringEntryArray EntriesVal)
```

VdxService

This section provides reference information for the VdxService interface.

getIRemoteVdxPort

Gets the stub for the remote service. The stub is a port of type IRemoteVdx.

Syntax: Here is the method signature:

```
IRemoteVdx getIRemoteVdxPort() throws javax.xml.rpc.ServiceException;
```

VersionVO

This section provides reference information on the VersionVO class.

getValue

Returns the version number of the service.

Syntax: Here is the method signature:

```
java.lang.String getValue()
```

VDX Example

The following code example shows how one might use the VDX service to access the attributes associated with entities defined in the directory abstraction layer. It demonstrates the use of ad hoc searches, as well as predefined searches called global queries. This code listing includes examples that use the getAttribute(), getAttributes(), query(), and globalQuery() methods on the service.

To get a reference to the SOAP endpoint for the VDX service, it calls a method called getVdxStub(). The implementation for this method is shown at the end of the listing:

NOTE: This example presumes that you have generated client stubs from the WSDL file `IRemoteVdx.wsdl`. Use the SOAP stack provider of your choice (such as AXIS or CFX) to generate client stubs.

With Apache CXF, for example, you should be able to generate the stubs to match the package names in the `import` statement by using the following command:

```
wsdl2java -p com.netiq.ws.client.vdx IRemoteVdx.wsdl
```

```
import java.util.Properties;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.xml.rpc.Stub;
import java.rmi.RemoteException;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.rmi.RemoteException;
import java.util.Calendar;
import java.util.Date;
import java.util.Hashtable;
import java.util.Map;
//
// Vdx imports
import com.netiq.ws.client.vdx.IRemoteVdx;
import com.netiq.ws.client.vdx.VdxService;
import com.netiq.ws.client.vdx.VdxServiceException;
import com.netiq.ws.client.vdx.VersionVO;
import com.netiq.ws.client.vdx.Attribute;
import com.netiq.ws.client.vdx.AttributeArray;
import com.netiq.ws.client.vdx.AttributeType;
import com.netiq.ws.client.vdx.ByteArrayArray;
import com.netiq.ws.client.vdx.BooleanArray;
```

```
import com.netiq.ws.client.vdx.DateArray;
import com.netiq.ws.client.vdx.StringArray;
import com.netiq.ws.client.vdx.IntegerArray;
import com.netiq.ws.client.vdx.EntryArray;
import com.netiq.ws.client.vdx.Entry;
import com.netiq.ws.client.vdx.EntityAttributeMap;


public class ServiceTest
{
  public static final int VDX           = 0;
  public static final int NOTIFICATION      = 1;
  public static final int RESOURCE         = 2;
  public static final int ENDPOINT_SERVICE  = VDX;

  private static final int LOCALHOST    = 0;  // localhost
  private static final int TESTSERVER      = 1;  // testserver
  private static final int SELECTED_URL     = TESTSERVER;

      private String [] SERVER_URLS = {
    "http://localhost:8080/IDMProv/vdx/service",
    "http://testserver:8080/IDMProv/vdx/service"
  };

  private String url = SERVER_URLS[SELECTED_URL];
  private String username = "cn=admin,ou=idmsample,o=netiq";
  private String password = "test";

      private String [] userAttributes = {
                  //"passwordAllowChange", // boolean
                  "UserPhoto",             // binary
                  //"loginTime",             // time
                  "Department",            // string
                  "Title",
                  "Email",
                  "manager",               // dn = string
                  "TelephoneNumber",
                  "directReports",
                  "FirstName",
                  //"surname",
                  "group",
                  "srvprvHideAttributes",
                  "NotificationPrefs",
                  "srvprvQueryList",
                  "Location",
                  };

  public ServiceTest() { };

  public static void main(String [] args)
  {
          ServiceTest serviceTest = new ServiceTest();
       //
          // Set default if no params are given
       int wService = ENDPOINT_SERVICE;
```

```
        if(args.length == 1)
               wService = Integer.parseInt(args[0]);

        try
        {
            serviceTest.run(wService);
        }
        catch(Exception e)
        {
        System.exit(-1);
        }
    }


    private void waitHere(long _time) { try { Thread.sleep(_time *
1000); } catch(InterruptedException ie) {} }


    public void run(int _service)
    throws Exception
    {
       if(_service == VDX)
       {
      System.out.println("Calling VDX endpoint");
      //
      // Get the version number
      getVersionTestCase();
      waitHere(2);
      //
      // Get attribute data for entity user
      getAttributeTestCase();
      waitHere(2);
      //
      // Get attributes
      getAttributesTestCase();
      waitHere(2);
      //
      // Query attributes
      queryAttributesTestCase();
      waitHere(2);
      //
      // Global query
             // Global query MUST be associated with a defined and
deployed query.
      // This can be done via the Designer.

      globalQueryTestCase();
       }
       else if(_service == NOTIFICATION)
       {
      System.out.println("Calling Notification endpoint");
      NotificationTest notificationTest = new
NotificationTest();
      //
      // Email Notification
```

```
    notificationTest.emailNotificationTestCase();
         }
     else if(_service == RESOURCE)
     {
    System.out.println("Calling Resource endpoint");
         }
         else
     {
    System.out.println("Unrecognized service selection");
         }
  }


public void globalQueryTestCase()
    throws Exception
    {

System.out.println("\n<=========queryAttributesTestCase=========>");
        try
        {
            //
            // Get the vdx stub
    IRemoteVdx vdxStub = getVdxStub(url, username, password);
            //
            // Create entry items corresponding to param key in DAL
            StringEntry [] entry = {
                    new StringEntry("titleattribute", "Chief Operating
Officer"),
                    new StringEntry("managerattribute",
"cn=jmiller,ou=users,ou=idmsample-pproto,o=netiq")
            };
            //
            // Create and set the array of entries (key,value pairs)
            StringEntryArray entryArr = new StringEntryArray();
            entryArr.setStringentry(entry);
            //
            // Create and set the map using the entries
            StringMap map = new StringMap();
            map.setEntries(entryArr);
            //
            // Define and execute the global query
            int QUERY_KEY_INDEX = 0;
            String [] queryKeyName = {"TestVdxGlobalQuery2",
"TestVdxGlobalQuery"};
            //
            // Results from global query TestVdxGlobalQuery2 ----->
cn=apalani,ou=users,OU=idmsample-pproto,O=netiq
            //
            // Make the vdx endpoint call
            StringArray array =
vdxStub.globalQuery(queryKeyName[QUERY_KEY_INDEX], map);
            String [] str = array.getString();
            if(str == null)
```

```
                throw new Exception("Global query returns null for key
name " + queryKeyName);
            else
            {
                System.out.println("Results for global query : " +
queryKeyName[QUERY_KEY_INDEX]);

System.out.println("================================================
===");
                for(int index = 0; index < str.length; index++)
                {
                    System.out.println(str[index]);
                }
            }
        }
        catch(VdxServiceException error)
        {
            System.out.println(error.getReason() );
            throw new Exception(error.getReason() );
        }
        catch(RemoteException error)
        {
            System.out.println(error.getMessage() );
            throw new Exception(error.getMessage() );
        }
    }


    public void queryAttributesTestCase()
    throws Exception
    {
    System.out.println("\nCalling queryAttributesTestCase() test
case");
    try
        {
    IRemoteVdx vdxStub = getVdxStub(url, username, password);

            StringArray attributes = new StringArray();
            attributes.setString(new String[]{"FirstName", "Title",
"UserPhoto", "Department"});
            String expression1 = "FirstName STARTWITH 'J'";
            String expression2 = "Title = 'Controller'";
            String expression3 = "vdxInteger > 0";
            String expression4 = "TelephoneNumber != '(555) 555-1201'";
            //
            // Test Cases
            // expression1 --> Should yield all users whose firstname
starts with J
            // expression1 AND expression2 --> Should yield jkelley who
is the Controller
            // expression1 AND expression3 --> Should yield only jmiller
            // expression1 AND expression4 --> Should yield all users
starting with J EXCEPT jmiller
            String finalExpression = expression1 + " AND " +
expression2;
```

```
            //
            // Make the vdx endpoint call
            EntityAttributeMap map = vdxStub.query("user", attributes,
finalExpression);
            EntryArray entryArray = map.getEntries();
            Entry [] entries = entryArray.getEntry();
            if(entries != null)
            {
                for(int index = 0; index < entries.length; index++)
                {
                    String dnKey = entries[index].getKey();
                    System.out.println("DN Key = " + dnKey);
                    AttributeArray attributeArray =
entries[index].getValues();
                    Attribute [] attributeData =
attributeArray.getAttribute();
                    for(int attrIndex = 0; attrIndex <
attributeData.length; attrIndex++)
                    {
                        //
                        // Determine how to handle the return data
                        examineAttributeData(attributeData[attrIndex],
" ");
                    }

                }
            }
        }
        catch(VdxServiceException error)
        {
            System.out.println(error.getReason() );
            throw new Exception(error.getReason() );
        }
        catch(RemoteException error)
        {
            System.out.println(error.getMessage() );
            throw new Exception(error.getMessage() );
        }
    }


    public void getVersionTestCase()
    throws Exception
    {
    System.out.println("\nCalling getVersionTestCase() test
case");

  try
  {
    IRemoteVdx vdxStub = getVdxStub(url, username, password);
            VersionVO version = vdxStub.getVersion();
            System.out.println("Version : " + version.getValue() );
  }
          catch(RemoteException error)
          {
```

```
                System.out.println(error.getMessage() );
                throw new Exception(error.getMessage() );
            }
        }


        public void getAttributeTestCase()
        throws Exception
        {
          System.out.println("\nCalling getAttributeTestCase() test
    case");

              try
              {
         IRemoteVdx vdxStub = getVdxStub(url, username, password);

                String recipient =
    "cn=jmiller,ou=users,ou=idmsample,o=netiq";
                String entity = "user";
                for(int attributeIndex = 0; attributeIndex <
    userAttributes.length; attributeIndex++)
                {
                    //
                    // Now, get the values for each attribute from the VDX
    layer
                    Attribute attributeData =
    vdxStub.getAttribute(recipient,
                                entity, userAttributes[attributeIndex]);
                    //
                    // Determine how to handle the return data
                    examineAttributeData(attributeData,
    userAttributes[attributeIndex]);
                }
             }
            catch(VdxServiceException error)
            {
                System.out.println(error.getReason() );
                throw new Exception(error.getReason() );
            }
            catch(RemoteException error)
            {
                System.out.println(error.getMessage() );
                throw new Exception(error.getMessage() );
            }
       }

        public void getAttributesTestCase()
        throws Exception
        {
          System.out.println("\nCalling getAttributesTestCase() test
    case");

              try
              {
         IRemoteVdx vdxStub = getVdxStub(url, username, password);
```

```
            String recipient =
"cn=jmiller,ou=users,ou=idmsample,o=netiq";
            String entity = "user";
    StringArray userAttributesArray = new
StringArray(userAttributes);
            AttributeArray attributeArray =
vdxStub.getAttributes(recipient,
      entity, userAttributesArray);
            Attribute [] attributeData = attributeArray.getAttribute();
            for(int index = 0; index < attributeData.length; index++)
            {
                //
                // Determine how to handle the return data
                examineAttributeData(attributeData[index],
userAttributes[index]);
            }
    }
        catch(VdxServiceException error)
        {
            System.out.println(error.getReason() );
            throw new Exception(error.getReason() );
        }
        catch(RemoteException error)
        {
            System.out.println(error.getMessage() );
            throw new Exception(error.getMessage() );
        }
   }


    private void examineAttributeData(Attribute _attribute, String
_attributeName)
    throws Exception
    {
        AttributeType type = _attribute.getType();
       System.out.println("Attribute type : " + type);
        //
        // What type are we dealing with?
        if(type.getValue().compareTo(AttributeType._Integer) == 0)
        {
            IntegerArray intArray = _attribute.getIntegers();
            int [] intData = intArray.getInt();
            if(intData == null)
                System.out.println(_attributeName + " attribute : " +
"null because no attribute value exists.");
            else
            {
                for(int intIndex = 0; intIndex < intData.length;
intIndex++)
                {
                    System.out.println(_attributeName + " attribute : "
+ intData[intIndex]);
                }
            }
```

```
        }
        else if(type.getValue().compareTo(AttributeType._Boolean) == 0)
        {
            BooleanArray boolArray = _attribute.getBooleans();
            boolean [] booleanData = boolArray.getBoolean();
            if(booleanData == null)
                System.out.println(_attributeName + " attribute : " +
"null because no attribute value exists.");
            else
            {
                for(int boolIndex = 0; boolIndex < booleanData.length;
boolIndex++)
                {
                    System.out.println(_attributeName + " attribute : "
+ booleanData[boolIndex]);
                }
            }
        }
        else if( (type.getValue().compareTo(AttributeType._String) ==
0) ||
                 (type.getValue().compareTo(AttributeType._DN) == 0) )
        {
            StringArray dataArray = _attribute.getStrings();
            String [] stringData = dataArray.getString();
            if(stringData == null)
                System.out.println(_attributeName + " attribute : " +
"null because no attribute value exists.");
            else
            {
                for(int strIndex = 0; strIndex < stringData.length;
strIndex++)
                {
                    System.out.println(_attributeName + " attribute : "
+ stringData[strIndex]);
                }
            }
        }
        else if(type.getValue().compareTo(AttributeType._Binary) == 0)
        {
            ByteArrayArray byteArray = _attribute.getBinaries();
            byte [][] byteData = byteArray.getBase64Binary();
            if(byteData == null)
                System.out.println(_attributeName + " attribute : " +
"null because no attribute value exists.");
            else
            {
                for(int byteIndex = 0; byteIndex < byteData.length;
byteIndex++)
                {
                    byte [] data = byteData[byteIndex];
                    //
                    // Save the data to a gif file and view it to
                    // make sure the binary return data is correct.
                    try
                    {
```

```
                        File fileObj = new File("C:\\temp\\photo.gif");
                        if(fileObj.exists())
                            fileObj.delete();
                        FileOutputStream fout = new
FileOutputStream(fileObj);
                        fout.write(data);
                        fout.flush();
                    }
                    catch(FileNotFoundException fne)
                    {
                        throw new Exception(fne.getMessage());
                    }
                    catch(IOException ioe)
                    {
                        throw new Exception(ioe.getMessage());
                    }
                }
            }
        }
        else if(type.getValue().compareTo(AttributeType._Time) == 0)
        {
            DateArray dateArray = _attribute.getDates();
            Calendar [] calendar = dateArray.getDatetime();
            if(calendar == null)
                System.out.println(_attributeName + " attribute : " +
"null because no attribute value exists.");
            else
            {
                for(int calIndex = 0; calIndex < calendar.length;
calIndex++)
                {
                    System.out.println(_attributeName + " attribute : "
+ calendar[calIndex].getTime().toString());
                }
            }
        }

    }


    /**
     * Method to obtain the remote interface to the Vdx endpoint
     * @param _url
     * @param _username
     * @param _password
     * @return IRemoteMetrics interface
     * @throws Exception
     */
    private IRemoteVdx getVdxStub(String _url, String _username, String
_password)
    throws Exception
    {
        Properties properties = new Properties();
        properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jnp.interfaces.NamingContextFactory");
```

```
        String lookup =
"xmlrpc:soap:com.netiq.ws.client.vdx.VdxService";

        InitialContext ctx = new InitialContext();
        VdxService svc = (VdxService) ctx.lookup(lookup);

        Stub stub = (Stub)svc.getIRemoteVdxPort();

        stub._setProperty(Stub.USERNAME_PROPERTY, _username);
        stub._setProperty(Stub.PASSWORD_PROPERTY, _password);
        stub._setProperty(Stub.SESSION_MAINTAIN_PROPERTY,
Boolean.TRUE);
        stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, _url);

        return (IRemoteVdx) stub;
    }

}
```

29 Role Web Service

This section describes the Role Web Service, which allows SOAP clients to access the role management and SoD management functions.

About the Role Web Service

To support access by third-party software applications, the Role subsystem includes a Web service endpoint called the Role Web Service. It supports a wide range of role management and SoD management functions.

This Appendix describes the programming interface for the Role Web Service.

Accessing the Test Page

You can access the Role Web Service endpoint using a URL similar to the following:

`http://server:port/warcontext/role/service?test`

For example, if your server is named “myserver”, your User Application is listening on port 8080, and your User Application war file is named “IDMPROV”, the URL would be:

`http://myserver:8080/IDMPROV/role/service?test`

You can also access the SOAP endpoint by going to the **Administration** within the User Application. To do this, you need to select the **Application Configuration** tab, then select **Web Services** from the left-navigation menu. After selecting **Web Services**, pick the Web Service endpoint you want from the list.

WARNING: The test page is disabled by default. Since some of the methods allow data to be updated, the test page presents a potential security vulnerability and should not be allowed in a production environment.

Servlet Declaration for the Test Page

A SOAP service using WSSDK is deployed by adding the following declarations in the deployment descriptor (i.e. WEB-INF/web.xml):

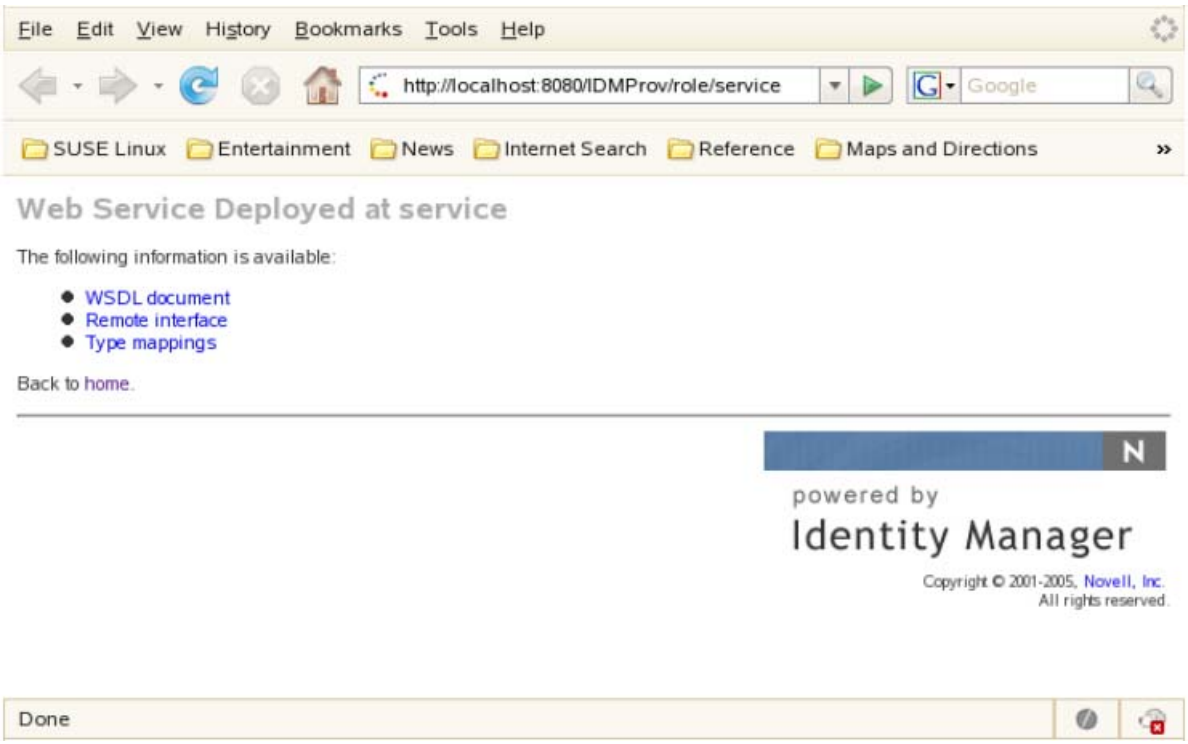
```
<servlet>
  <servlet-name>Role</servlet-name>
  <servlet-
class>com.novell.idm.nrf.soap.ws.role.impl.RoleServiceSkeletonImpl</
servlet-class>
```

```
<servlet-mapping>
  <servlet-name>Role</servlet-name>
  <url-pattern>/role/service</url-pattern>
</servlet-mapping>
</servlet>
```

This follows the normal servlet declaration pattern. It indicates that the servlet com.novell.idm.nrf.soap.ws.role.impl.RoleServiceSkeletonImpl is deployed at /role/service.

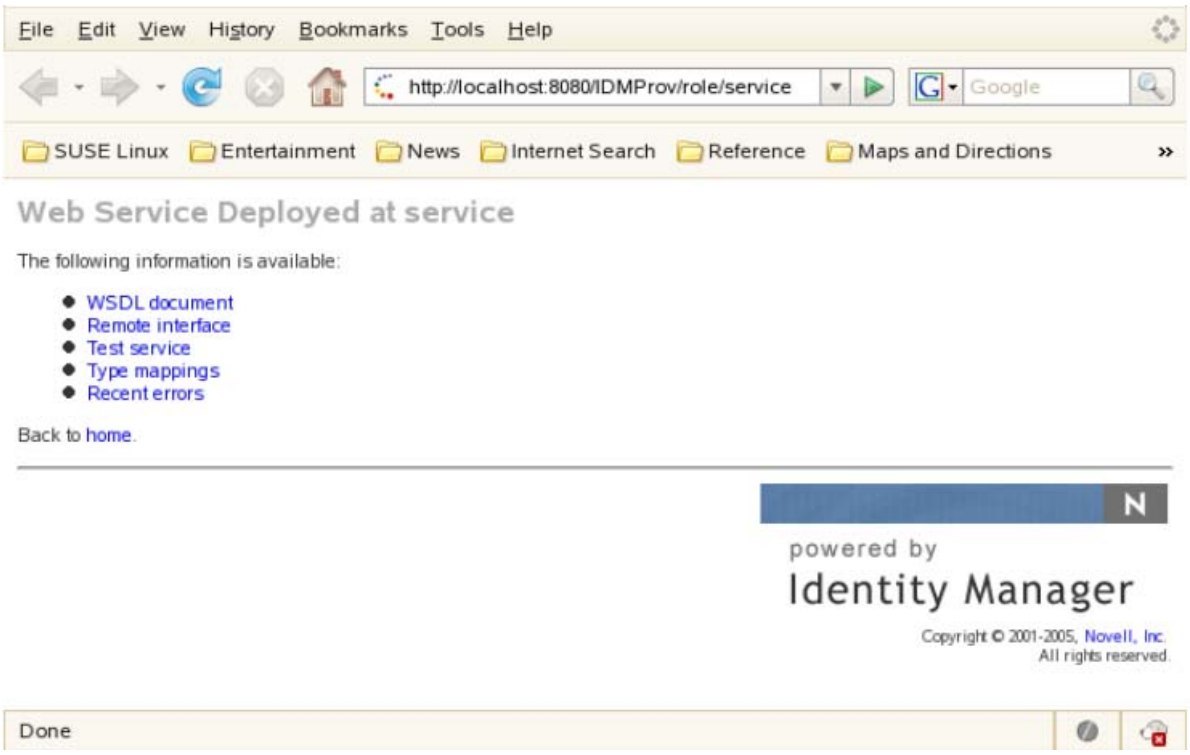
When a user reaches this servlet using a HTTP GET by entering `http://server-name/context/role/service` (for example, `http://localhost:8080/IDMProv/role/service`) in their browser, the WSSDK provides a page that exposes some information about the deployed service. By default the page looks like this:

Figure 29-1 *SOAP Service with Test Page Disabled*



After you enable the test page, the **Test Service** link is available:

Figure 29-2 *SOAP Servlet with Test Page Enabled*



On the test page, the user can retrieve the WSDL document that describes the Web Service, see the Java Remote Interface that represents the service, and also see the type mappings from XML to Java. In addition, the user can test the service by invoking individual methods.

Enabling the Test Page

WARNING: The test page is disabled by default. Since some of the methods allow data to be updated, the test page presents a potential security vulnerability and should not be allowed in a production environment.

To enable the test page, you need to update the WEB-INF/web.xml file in the IDMProv.war file. Before you make your changes, the web.xml should look like this:

```
<servlet>
  <servlet-name>Role</servlet-name>
  <servlet-
class>com.novell.idm.nrf.soap.ws.role.impl.RoleServiceSkeletonImpl</
servlet-class>
  <init-param>
    <param-name>com.novell.soa.ws.test.disable</param-name>
    <param-value>true</param-value>
  </init-param>
</servlet>
```

Change the servlet declaration, as follows:

```
<servlet>
  <servlet-name>Role</servlet-name>
  <servlet-
class>com.novell.idm.nrf.soap.ws.role.impl.RoleServiceSkeletonImpl</
servlet-class>
</servlet>
```

Accessing the WSDL

You can access the WSDL for the Role Web Service using a URL similar to the following:

`http://server:port/warcontext/role/service?wsdl`

For example, if your server is named “myserver”, your User Application is listening on port 8080, and your User Application war file is named “IDMPROV”, the URL would be:

`http://myserver:8080/IDMPROV/role/service?wsdl`

Generating the Stub Classes

Before using the Web Service, you need to use the WSSDK tool or another SOAP tool kit to generate the stub classes. To allow your code to find the stub classes, you also need to add the JAR that contains the stub classes to your classpath.

If you want to use the NetIQ WSSDK tool, you can generate the client stubs by extracting the WSDL and running the wsdl2java utility. For example, you could run this command to generate the stubs in a package called com.novell.soa.af.role.soap.impl:

```
"C:\Program Files\Java\jdk1.6.0_31\bin\java" -cp "../lib/wssdk.jar;../lib/
jaxrpc-api.jar";"../lib/mail.jar";"../lib/activation.jar";"c:\Program
Files\Java\jdk1.6.0_31\lib\tools.jar";
com.novell.soa.ws.impl.tools.wsdl2java.Main -verbose -ds gensrc -d C:\ -
noskel -notie -genclient -keep -package com.novell.soa.af.role.soap.impl -
javadoc role.wsdl
```

You can change the wsdl2java parameters to suit your requirements.

Removing Administrator Credential Restrictions

The Role Web Service supports two levels of security, one that restricts access to Role Administrators, and another that restricts access to the authenticated user. The default setting restricts access to all operations to the Role Administrator.

You can modify the settings for security configuration in the `ism-configuration.properties` file, located by default in the `/netiq/idm/apps/tomcat/conf` directory. . Each property can be set to true or false. A value of true locks down the operation, whereas a value of false opens up the operation.

You can open up the Role Web Service to authenticated users by setting the RoleService/Role/soap property to false. To open up a particular operation to authenticated users, you need to set the property for that operation (RoleService/Role/soap/*operation*) to false as well. If you set all of the properties to false, you can open up all operations to authenticated users. The *operation* names are the same as the names of the methods supported by the service.

Example To ensure that the security configuration opens up all operations within the Role Web Service, the `ism-configuration.properties` file must have the following setting:

```
RoleService/Role/soap = false
```

Role API

This section provides details about the methods available with the Role Web service. This API presumes you're using Java code generated by the WSSDK toolkit. The API will be different if you're using another Web Service toolkit.

IRemoteRole

This section provides reference information for each method associated with the IRemoteRole interface.

createResourceAssociation

Create a resource association and return the resource association object with the newly created resource association DN.

Syntax: Here is the method signature:

```
ResourceAssociation
createResourceAssociation(com.novell.idm.nrf.soap.ws.ResourceAssociation
resourceAssociation)
        throws com.novell.idm.nrf.soap.ws.NrfServiceException,
java.rmi.RemoteException;
```

deleteResourceAssociation

Deletes a resource association object.

Syntax: Here is the method signature:

```
void deleteResourceAssociation(com.novell.idm.nrf.soap.ws.DNString
resourceAssociationDn)
        throws com.novell.idm.nrf.soap.ws.NrfServiceException,
java.rmi.RemoteException;
```

getResourceAssociations

Retrieves resource association objects for a given role DN or resource DN. If the *roleDn* and *resourceDn* parameters are null, the entire list is returned.

Syntax: Here is the method signature:

```
ResourceAssociation[]
getResourceAssociations(com.novell.idm.nrf.soap.ws.DNString roleDn,
com.novell.idm.nrf.soap.ws.DNString resourceDn)
        throws com.novell.idm.nrf.soap.ws.NrfServiceException,
java.rmi.RemoteException;
```

Create Role

Creates a new role according to the specified parameters and returns the DN of the created role.

A correlation ID is generated automatically for this method that uses this format:

```
UserApp#RemoteRoleRequest#xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

The correlation ID is used for auditing.

Syntax: Here is the method signature:

```
public DNString createRole(RoleRequest role)
            throws NrfServiceException, RemoteException;
```

createRoleAid

Creates a new role with a correlation ID that you provide. The correlation ID is used for auditing to link a set of related roles. This method returns the DN of the created role.

Syntax: Here is the method signature:

```
public DNString createRoleAid (RoleRequest role, String correlationId)
            throws NrfServiceException, RemoteException;
```

findRoleByExampleWithOperator

Finds an array of Role objects based on the search criteria specified in the given Role object. This method also lets you specify whether to use AND as the operator for multi-value searches.

Syntax: Here is the method signature:

```
RoleArray findRoleByExampleWithOperator(Role searchCriteria, boolean
useAndForMultiValueSearch) throws NrfServiceException,
java.rmi.RemoteException
```

This method follows a query by example approach. It allows you to populate a Role object to specify the desired search criteria. An AND operation is always used across multiple attributes within the Role search object. For example, you might provide a value for the `name` and `description` attributes, which indicates that the criteria for both attributes must be satisfied for a successful search.

The second parameter (useAndForMultiValueSearch) allows you to specify which operator should be used for multi-valued attributes (such as when multiple child roles are provided). A value of true indicates that AND should be used for these operations, whereas a value of false indicates that OR should be used.

Not all attributes in the Role object can be used for the search expression. Values found in the non-supported search attributes are ignored.

Table 29-1 *Guidelines for Defining Search Criteria in the Role Object*

Attribute	Supported?	Description

approvers	Yes	Uses a standard LDAP equal operator for the search. You can enter multiple approvers and use the operator parameter to determine whether an AND or an OR is used for the multi-valued search. You need to provide valid Dns for the approvers. Note that an approver is made up of multiple parts. It is of type TypedNameSyntax. You need to specify the sequence number of the approver to execute a successful search. This is a limitation in LDAP.
		
		Sample SOAP Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:approvers>`
		`<!--Zero or more repetitions:-->`
		`<ser:approver>`
		`<ser:approverDN>cn=ablake,ou=users,ou=medical-idmsample,o=netiq</ser:approverDN>`
		`<ser:sequence>1</ser:sequence>`
		`</ser:approver>`
		`</ser:approvers>`
		`</ser:role>`
		`<ser:operator>false</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The example above shows how to find roles that have the specified approver associated with them. An OR search is used since the operator parameter is set to false.

Attribute	Supported?	Description

childRoles	Yes	Uses a standard LDAP equal operator for the search. You can enter multiple child roles and use the operator parameter to determine whether an AND or an OR is used for the multi-valued search. You need to provide valid Dns for the child roles.
		
		Sample SOAP Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:childRoles>`
		`<!--Zero or more repetitions:-->`
		`<ser:dnstring>`
		`<ser:dn>cn=Doctor,cn=Level20,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=PicassoDriver,cn=TestDrivers,o=netiq</ser:dn>`
		`</ser:dnstring>`
		`<ser:dnstring>`
		`<ser:dn>cn=Nurse,cn=Level20,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=PicassoDriver,cn=TestDrivers,o=netiq</ser:dn>`
		`</ser:dnstring>`
		`</ser:childRoles>`
		`</ser:role>`
		`<ser:operator>false</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The example above shows how to find roles with a child role of “Doctor” or “Nurse. An OR search is used since the operator parameter is set to false.

description	Yes	Uses an LDAP contains search. All entries are prefixed and suffixed with the * (wild card character). Therefore, a search for “Doctor” translates to “*Doctor*”. This is to accommodate searches across any localized language.
		
		Sample SOAP Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:description>Doctor</ser:description>`
		`</ser:role>`
		`<ser:operator>false</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The example above shows how to find roles with a description of “Doctor”. This description string results in a search string of “*Doctor*”.

Attribute	Supported?	Description

entityKey	Yes	If entered, this attribute causes a getRole operation to be performed. All other search criteria are ignored in this case.
		
		Sample SOAP Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:entityKey>cn=Doctor,cn=Level20,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=PicassoDriver,cn=TestDrivers,o=netiq</ser:entityKey>`
		`</ser:role>`
		`<ser:operator>false</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The example above shows how to retrieve a role with a specific entity key.

implicitContainers	Yes	Uses a standard LDAP equal operator for the search. You can enter multiple implicit containers and use the operator parameter to determine whether an AND or an OR will be used for the multi-valued search. You need to provide valid Dns for the implicit containers.
		
		Sample SOAP Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:implicitContainers>`
		`<!--Zero or more repetitions:-->`
		`<ser:dnstring>`
		`<ser:dn>ou=medical-idmsample,o=netiq</ser:dn>`
		`</ser:dnstring>`
		`</ser:implicitContainers>`
		`</ser:role>`
		`<ser:operator>false</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The example above shows how to find roles that have the specified implicit container associated with them. An OR search is used since the operator parameter is set to false.

Attribute	Supported?	Description

implicitGroups	Yes	Uses a standard LDAP equal operator for the search. You can enter multiple implicit groups and use the operator parameter to determine whether an AND or an OR will be used for the multi-valued search. You need to provide valid Dns for the implicit groups.
		
		Sample SOAP Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:implicitGroups>`
		`<!--Zero or more repetitions:-->`
		`<ser:dnstring>`
		`<ser:dn>cn=HR,ou=groups,ou=medical-idmsample,o=netiq</ser:dn>`
		`</ser:dnstring>`
		`</ser:implicitGroups>`
		`</ser:role>`
		`<ser:operator>false</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The example above shows how to find roles that have the specified implicit group associated with them. An OR search is used since the operator parameter is set to false.

name	Yes	Uses an LDAP contains search. All entries will be prefixed and suffixed with the * (wild card character). Therefore, a search for “Doctor” translates to “*Doctor*”. This is to accommodate searches across any localized language.
		
		Sample SOAP Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:name>Doctor</ser:name>`
		`</ser:role>`
		`<ser:operator>false</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The above example shows how to find roles with a name of “Doctor”. The name string results in a search string of “*Doctor*”.

Attribute	Supported?	Description

owners	Yes	Uses a standard LDAP equal operator for the search. You can enter multiple owners and use the operator parameter to determine whether an AND or an OR is used for the multi-valued search. You must provide valid Dns for the owners.
		
		SoapUI Example Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:owners>`
		`<!--Zero or more repetitions:-->`
		`<ser:dnstring>`
		`<ser:dn>cn=ablake,ou=users,ou=medical-idmsample,o=netiq</ser:dn>`
		`</ser:dnstring>`
		`<ser:dnstring>`
		`<ser:dn>cn=mmackenzie,ou=users,ou=medical-idmsample,o=netiq</ser:dn>`
		`</ser:dnstring>`
		`</ser:owners>`
		`</ser:role>`
		`<ser:operator>true</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The example above shows how to find roles that have the specified owners. An AND search is used since the operator parameter is set to true.

parentRoles	Yes	Uses a standard LDAP equal operator for the search. You can enter multiple parent roles and use the operator parameter to determine whether an AND or an OR is used for the multi-valued search. You must provide valid Dns for the parent roles.
		
		Sample SOAP Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:parentRoles>`
		`<!--Zero or more repetitions:-->`
		`<ser:dnstring>`
		`<ser:dn>cn=Doctor-East,cn=Level30,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=PicassoDriver,cn=TestDrivers,o=netiq</ser:dn>`
		`</ser:dnstring>`
		`<ser:dnstring>`
		`<ser:dn>cn=Doctor-West,cn=Level30,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=PicassoDriver,cn=TestDrivers,o=netiq</ser:dn>`
		`</ser:dnstring>`
		`</ser:parentRoles>`
		`</ser:role>`
		`<ser:operator>true</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The example above shows how to find roles that have the specified parent roles. An AND search is used since the operator parameter is set to true.

Attribute	Supported?	Description

quorum	Yes	Uses a standard LDAP equal operator for the search.
		
		Sample SOAP Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:quorum>50%</ser:quorum>`
		`</ser:role>`
		`<ser:operator>false</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The example above shows how to find roles with the specified quorum search string. The search string can include the wild card character (“*”).

requestDef	Yes	Uses a standard LDAP equal operator for the search. You must provide a valid DN for the request definition.
		
		Sample SOAP Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:requestDef>cn=Role Approval,cn=RequestDefs,cn=AppConfig,cn=PicassoDriver,cn=TestDrivers,o=netiq</ser:requestDef>`
		`</ser:role>`
		`<ser:operator>false</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The example above shows how to find roles with the specified request definition DN.

roleCategoryKeys	Yes	Uses a standard LDAP equal operator for the search. You can enter multiple category keys and use the operator parameter to determine whether an AND or an OR is used for the multi-valued search.
		
		Sample SOAP Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:roleCategoryKeys>`
		`<!--Zero or more repetitions:-->`
		`<ser:categorykey>`
		`<ser:categoryKey>doctor</ser:categoryKey>`
		`</ser:categorykey>`
		`<ser:categorykey>`
		`<ser:categoryKey>nurse</ser:categoryKey>`
		`</ser:categorykey>`
		`</ser:roleCategoryKeys>`
		`</ser:role>`
		`<ser:operator>false</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The example above shows how to find roles with a category of “doctor” or “nurse. An OR search is used since the operator parameter is set to false.

Attribute	Supported?	Description

roleLevel	Yes	Uses a standard LDAP equal operator for the search. You can only enter one level at a time.
		
		Sample SOAP Request:
		
		`<ser:findRoleByExampleWithOperatorRequest>`
		`<ser:role>`
		`<ser:roleLevel>`
		`<ser:level>10</ser:level>`
		`</ser:roleLevel>`
		`</ser:role>`
		`<ser:operator>false</ser:operator>`
		`</ser:findRoleByExampleWithOperatorRequest>`
		
		The example above shows how to find all level 10 roles.

associatedRoles	No	Not supported.
entitlementRef	No	Not supported.
roleAssignments	No	Not supported.
systemRole	No	Not supported.

findSodByExample

Finds all SoD objects based on the search criteria in the given SOD object.

Syntax: Here is the method signature:

```
SodArray findSodByExample(Sod sod) throws NrfServiceException,
java.rmi.RemoteException
```

findSodByExampleWithOperator

Finds all SoD objects based on the search criteria found in the given SOD object. This method also lets you specify whether to use And as the operator for multi-value searches.

Syntax: Here is the method signature:

```
SodArray findSodByExampleWithOperator(Sod searchCriteria, boolean
useAndForMultiValueSearch) throws NrfServiceException,
java.rmi.RemoteException
```

findSodById

Find by key.

Syntax: Here is the method signature:

```
Sod findSodById(java.lang.String entityKey) throws NrfServiceException,
java.rmi.RemoteException
```

getAssignedIdentities

Returns returns the list of identities having a particular role DN.

Syntax: Here is the method signature:

```
RoleAssignment[] getAssignedIdentities(java.lang.String roleDN,
IdentityType identityType, boolean directAssignOnly)
```

getConfigProperty

Retrieves configuration properties stored in the User Application configuration XML files by passing in a configuration property key or macro name.

Syntax: Here is the method signature:

```
public ConfigProperty getConfigProperty(String configPropertyKey) throws
NrfServiceException, RemoteException;
```

The configPropertyKey parameter can accept a fully qualified configuration key name from any of the configuration XML files, such as the following:

```
DirectoryService/realms/jndi/params/USER_ROOT_CONTAINER
```

Alternativelly, the configPropertyKey parameter can accept a macro name that references a fully qualified configuration key name. The following macro names are allowed:

Table 29-2 *Macro Names Allowed*

Configuration Macro Name	Configuration Key Value
USER_CONTAINER	DirectoryService/realms/jndi/params/ USER_ROOT_CONTAINER
GROUP_CONTAINER	DirectoryService/realms/jndi/params/ GROUP_ROOT_CONTAINER
ROOT_CONTAINER	DirectoryService/realms/jndi/params/ROOT_NAME
PROVISIONING_DRIVER	DirectoryService/realms/jndi/params/ PROVISIONING_ROOT

getConfiguration

Returns the role system configuration defined in the Role Catalog root (nrfConfiguration).

Syntax: Here is the method signature:

```
Configuration getConfiguration() throws NrfServiceException,
java.rmi.RemoteException
```

getContainer

Gets container and role information for a given container DN.

Syntax: Here is the method signature:

```
Container getContainer(java.lang.String containerDn)
throws NrfServiceException, java.rmi.RemoteException
```

getExceptionList

Returns a list of Sod instances for all SOD violations found for a specific identity and type.

Syntax: Here is the method signature:

```
SodArray getExceptionsList(java.lang.String identity, IdentityType
identityType) throws NrfServiceException, java.rmi.RemoteException
```

getGroup

Gets group and role information for a given group DN.

Syntax: Here is the method signature:

```
Group getGroup(java.lang.String groupDn) throws NrfServiceException,
java.rmi.RemoteException
```

getIdentitiesInViolation

Returns a map of identities which are in violation of a given SoD.

Syntax: Here is the method signature:

```
IdentityTypeDnMapArray getIdentitiesInViolation(java.lang.String sodDn)
throws NrfServiceException, java.rmi.RemoteException
```

getIdentityRoleConflicts

Returns a list of Sod instances for all SOD conflicts found for a given list of roles for a given identity.

Syntax: Here is the method signature:

```
SodArray getIdentityRoleConflicts(java.lang.String identity, IdentityType
identityType, DNStringArray requestedRoles) throws NrfServiceException,
java.rmi.RemoteException
```

getRole

Retrieves a role object defined by a role DN. Returns several role attributes, such as name, dn, description, role level. Returns child roles, assigned containers, and assigned groups. However, this API does not return assigned users. If you want assigned users, use the getAssignedIdentities API with USER for identityType and true for directAssignOnly.

Syntax: Here is the method signature:

```
Role getRole(java.lang.String roleDn) throws NrfServiceException,
java.rmi.RemoteException
```

getRoleAssignmentRequestStatus

Returns a list of role assignment request status instances given a correlation ID.

Syntax: Here is the method signature:

```
RoleAssignmentRequestStatusArray
getRoleAssignmentRequestStatus(java.lang.String correlationId) throws
NrfServiceException, java.rmi.RemoteException
```

getRoleAssignmentRequestStatusByIdentityType

Returns a list of role assignment request status instances given an identity and an identity type.

Syntax: Here is the method signature:

```
RoleAssignmentRequestStatusArray
getRoleAssignmentRequestStatusByIdentityType(java.lang.String identityDn,
IdentityType identityType) throws NrfServiceException,
java.rmi.RemoteException
```

getRoleAssignmentTypeInfo

Retrieves details about a RoleAssignmentType.

Syntax: Here is the method signature:

```
RoleAssignmentTypeInfo getRoleAssignmentTypeInfo(RoleAssignmentType type)
throws NrfServiceException, java.rmi.RemoteException
```

getRoleCategories

Gets role categories.

Syntax: Here is the method signature:

```
CategoryArray getRoleCategories() throws NrfServiceException,
java.rmi.RemoteException
```

getRoleConflicts

Returns a list of Sod instances found for all given roles. This method always returns a list.

Syntax: Here is the method signature:

```
SodArray getRoleConflicts(DNStringArray roles) throws NrfServiceException,
java.rmi.RemoteException
```

getRoleLevels

Gets the role levels.

Syntax: Here is the method signature:

```
RoleLevelArray getRoleLevels() throws NrfServiceException,
java.rmi.RemoteException
```

getRoleLocalizedStrings

Gets role localized strings, such as names and descriptions. The method takes an integer parameter that allows you to specify the type of the string. The number 1 indicates names; the number 2 indicates descriptions.

Syntax: Here is the method signature:

```
public LocalizedValue[] getRoleLocalizedStrings(DNString roleDn, int type)
            throws NrfServiceException, RemoteException;
```

getRolesInfo

Returns a list of RoleInfo instances given a list of role DNs.

Syntax: Here is the method signature:

```
RoleInfoArray getRolesInfo(DNStringArray roleDns) throws
NrfServiceException, java.rmi.RemoteException
```

getRolesInfoByCategory

Returns a list of RoleInfo instances given a list of role category keys.

Syntax: Here is the method signature:

```
RoleInfoArray getRolesInfoByCategory(CategoryKeyArray roleCategoryKeys)
throws NrfServiceException, java.rmi.RemoteException
```

getRolesInfoByLevel

Returns a list of RoleInfo instances given a list of role levels.

Syntax: Here is the method signature:

```
RoleInfoArray getRolesInfoByLevel(LongArray roleLevels) throws
NrfServiceException, java.rmi.RemoteException
```

getTargetSourceConflicts

Returns a list of Sod instances for all SOD conflicts defined between the target role DN and the source role DN.

Syntax: Here is the method signature:

```
SodArray getTargetSourceConflicts(java.lang.String targetName,
java.lang.String sourceName) throws NrfServiceException,
java.rmi.RemoteException
```

getUser

Gets user info including all role assignments for a given user DN stored in a UserIdentity object.

Syntax: Here is the method signature:

```
User getUser(java.lang.String userDn) throws NrfServiceException,
java.rmi.RemoteException
```

getVersion

Returns the version of this Web Service.

Syntax: Here is the method signature:

```
VersionVO getVersion() throws java.rmi.RemoteException
```

isUserInRole

Returns boolean flag; true if role has been assigned to a User identity.

Syntax: Here is the method signature:

```
boolean isUserInRole(java.lang.String userDn, java.lang.String roleDn)
```

modifyRole

Modifies a role definition. This method does not update localized strings. Use the getRoleLocalizedStrings(DNString roleDn, LocalizedString[] locStrings, int strType) method to update localized names or descriptions for a role.

A correlation ID is generated automatically for this method that uses this format:

```
UserApp#RemoteRoleRequest#xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

The correlation ID is used for auditing.

Syntax: Here is the method signature:

```
public Role modifyRole(Role role)
            throws NrfServiceException, RemoteException;
```

modifyRoleAid

Modifies a role definition with a correlation ID that you provide. The correlation ID is used for auditing to link a set of related roles. This method does not update localized strings. Use the getRoleLocalizedStrings(DNString roleDn, LocalizedString[] locStrings, int strType) method to update localized names or descriptions for a role.

Syntax: Here is the method signature:

```
public Role modifyRoleAid(Role role, String correlationId)
            throws NrfServiceException, RemoteException;
```

removeRoles

Deletes specified roles from the Role Catalog and returns an array of DNs for the deleted roles as a confirmation.

A correlation ID is generated automatically for this method that uses this format:

```
UserApp#RemoteRoleRequest#xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

The correlation ID is used for auditing.

Syntax: Here is the method signature:

```
public DNString[] removeRoles(DNString[] roleDns)
            throws NrfServiceException, RemoteException;
```

removeRolesAid

Deletes specified roles from the Role Catalog with a correlation ID that you provide. The correlation ID is used for auditing to link a set of related roles. This method returns an array of DNs for the deleted roles as a confirmation.

Syntax: Here is the method signature:

```
public DNString[] removeRolesAid(DNString[] roleDns, String correlationId)
            throws NrfServiceException, RemoteException;
```

requestRolesAssignment

Returns a list of request DNs created by the role assignment. Be aware that the role assignment expires only if the role is assigned to a user and not when it is assigned to a group or a container.

If you do not want to supply date (effective or expiration) for role assignments with the requestRolesAssignment endpoint, then you must remove these two elements from the SOAP call. They must not be included with empty tags:

```
<ser:effectiveDate/>
<ser:expirationDate/>
```

If you want to omit the effective date or the expiration date, a request similar to the following will work:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
"
xmlns:ser="http://www.netiq.com/role/service">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:requestRolesAssignmentRequest>
         <!--Optional:-->
         <ser:assignRequest>
            <ser:actionType>grant</ser:actionType>
            <ser:assignmentType>USER_TO_ROLE</ser:assignmentType>
            <ser:correlationID>testpolina</ser:correlationID>
            <ser:identity>cn=uaadmin,ou=sa,o=data</ser:identity>
            <ser:originator/>
            <ser:reason>test without expiration date</ser:reason>
            <ser:roles>
               <!--Zero or more repetitions:-->
               <ser:dnstring>
                  <ser:dn>cn=test2
id,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=User Application
Driver,cn=driverset1,o=system</ser:dn>
               </ser:dnstring>
            </ser:roles>
            <ser:sodOveridesRequested/>
         </ser:assignRequest>
      </ser:requestRolesAssignmentRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

With that said, without the these two elements in the soap request, the request will not validate. It will work, but will not validate.

Syntax: Here is the method signature:

```
DNStringArray requestRolesAssignment(RoleAssignmentRequest
roleAssignmentRequest) throws NrfServiceException,
java.rmi.RemoteException
```

setRoleLocalizedStrings

Sets role localized strings, such as names and descriptions.

A correlation ID is generated automatically for this method that uses this format:

```
UserApp#RemoteRoleRequest#xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

The correlation ID is used for auditing.

Syntax: Here is the method signature:

```
public LocalizedValue[] setRoleLocalizedStrings(DNString roleDn,
LocalizedValue[] locStrings, int type)
            throws NrfServiceException, RemoteException;
```

setRoleLocalizedStringsAid

Sets role localized strings, such as name and description, with a correlation ID that you provide. The correlation ID is used for auditing to link a set of related roles.

Syntax: Here is the method signature:

```
public LocalizedValue[] setRoleLocalizedStringsAid(DNString roleDn, String
correlationId, LocalizedValue[] locStrings, int type)
            throws NrfServiceException, RemoteException;
```

Approver

Class to hold the approver information for SOD or normal request approvals.

Approver constructors

The Approver class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
Approver()
```

getApproverDN

Gets the approver DN.

Syntax: Here is the method signature:

```
public java.lang.String getApproverDN()
```

getSequence

Gets the approver sequence.

Syntax: Here is the method signature:

```
public long getSequence()
```

setApproverDN

Sets the approver DN.

Syntax: Here is the method signature:

```
public void setApproverDN(java.lang.String approverDN)
```

setSequence

Sets the approver sequence.

Syntax: Here is the method signature:

```
public void setSequence(long sequence)
```

ApproverArray

This section provides reference information on the ApproverArray class.

ApproverArray constructors

The ApproverArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
ApproverArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of Attribute objects as a parameter:

```
ApproverArray(Approver[] ApproverVal)
```

getApprover

Returns an array of Approver objects.

Syntax: Here is the method signature:

```
Approver[] getApprover()
```

setApprover

Sets the array of Approver objects associated with the ApproverArray class.

Syntax: Here is the method signature:

```
void setApprover (Approver[] ApproverVal)
```

Category

Class to represent a role category.

Category constructors

The Category class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
Category()
```

getCategoryKey

Gets the category key.

Syntax: Here is the method signature:

```
public java.lang.String getCategoryKey()
```

getCategoryLabel

Gets the category label.

Syntax: Here is the method signature:

```
public java.lang.String getCategoryLabel()
```

setCategoryKey

Sets the category key.

Syntax: Here is the method signature:

```
public void setCategoryKey(java.lang.String categoryKey)
```

setCategoryLabel

Sets the category label.

Syntax: Here is the method signature:

```
public void setCategoryLabel(java.lang.String categoryLabel)
```

CategoryArray

This section provides reference information on the CategoryArray class.

CategoryArray constructors

The CategoryArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
CategoryArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of Category objects as a parameter:

```
CategoryArray(Category[] CategoryVal)
```

getCategory

Returns an array of Category objects.

Syntax: Here is the method signature:

```
Category[] getCategory()
```

setCategory

Sets the array of Category objects associated with the CategoryArray class.

Syntax: Here is the method signature:

```
void setCategory(Category[] CategoryVal)
```

CategoryKey

Class to hold a Category Key.

CategoryKey constructors

The CategoryKey class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
CategoryKey()
```

Syntax 2: Here is the syntax for a constructor that takes a String as a parameter:

```
CategoryKey(java.lang.String categoryKey)
```

getCategoryKey()

Gets the categoryKey.

Syntax: Here is the method signature:

```
public java.lang.String getCategoryKey()
```

setCategoryKey

Sets the category key.

Syntax: Here is the method signature:

```
public void setCategoryKey(java.lang.String categoryKey)
```

CategoryKeyArray

This section provides reference information on the CategoryKeyArray class.

CategoryKeyArray constructors

The CategoryKeyArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
CategoryKeyArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of CategoryKey objects as a parameter:

```
CategoryKeyArray(CategoryKey[] CategoryVal)
```

getCategorykey

Returns an array of Category objects.

Syntax: Here is the method signature:

```
CategoryKey[] getCategorykey()
```

setCategorykey

Sets the array of CategoryKey objects associated with the CategoryKeyArray class.

Syntax: Here is the method signature:

```
void setCategorykey(CategoryKey[] CategoryKeyVal)
```

Configuration

Class to represent the configuration object.

Configuration constructors

The Configuration class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
Configuration()
```

getDefaultRequestDef

Gets the default request definition.

Syntax: Here is the method signature:

```
public java.lang.String getDefaultRequestDef()
```

getDefaultSODRequestDef

Gets the default SOD request definition.

Syntax: Here is the method signature:

```
public java.lang.String getDefaultSODRequestDef()
```

getRemovalGracePeriod

Gets the removal grace period.

Syntax: Here is the method signature:

```
public int getRemovalGracePeriod()
```

getReportContainer

Gets the report container.

Syntax: Here is the method signature:

```
public java.lang.String getReportContainer()
```

getRoleLevels

Gets the role levels.

Syntax: Here is the method signature:

```
public RoleLevelArray getRoleLevels()
```

getRoleRequestContainer

Gets the role request container.

Syntax: Here is the method signature:

```
public java.lang.String getRoleRequestContainer()
```

getRolesContainer

Gets the role container.

Syntax: Here is the method signature:

```
public java.lang.String getRolesContainer()
```

getSODApprovers

Gets SOD approvers.

Syntax: Here is the method signature:

```
public ApproverArray getSODApprovers()
```

getSODContainer

Gets the SOD container.

Syntax: Here is the method signature:

```
public java.lang.String getSODContainer()
```

getSODQuorum

Gets the SOD quorum amount.

Syntax: Here is the method signature:

```
public java.lang.String getSODContainer()
```

getSODRequestDef

Gets the SOD request definition.

Syntax: Here is the method signature:

```
public java.lang.String getSODRequestDef()
```

setDefaultRequestDef

Sets the default request definition.

Syntax: Here is the method signature:

```
public void setDefaultRequestDef(java.lang.String defaultRequestDef)
```

setDefaultSODRequestDef

Sets the default SOD request definition.

Syntax: Here is the method signature:

```
public void setDefaultSODRequestDef(java.lang.String defaultSODRequestDef)
```

setRemovalGracePeriod

Sets the removal grace period.

Syntax: Here is the method signature:

```
public void setRemovalGracePeriod(int removalGracePeriod)
```

setReportContainer

Sets the report container.

Syntax: Here is the method signature:

```
public void setReportContainer(java.lang.String reportContainer)
```

setRoleLevels

Sets the role levels.

Syntax: Here is the method signature:

```
public void setRoleLevels(RoleLevelArray roleLevels)
```

setRoleRequestContainer

Sets the role request container.

Syntax: Here is the method signature:

```
public void setRoleRequestContainer(java.lang.String roleRequestContainer)
```

setRolesContainer

Sets the role container.

Syntax: Here is the method signature:

```
public void setRolesContainer(java.lang.String rolesContainer)
```

setSODApprovers

Sets the SoD approvers.

Syntax: Here is the method signature:

```
public void setSODApprovers(ApproverArray sODApprovers)
```

setSODContainer

Sets the SoD container.

Syntax: Here is the method signature:

```
public void setSODContainer(java.lang.String sODContainer)
```

Container

Class to represent a Container object.

Container constructors

The Container class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
Container()
```

getAssociatedRoles

Gets associated roles for this identity.

Syntax: Here is the method signature:

```
public DNStringArray getAssociatedRoles()
```

getEntityKey

Gets identity entity key.

Syntax: Here is the method signature:

```
public java.lang.String getEntityKey()
```

getIdentityType

Gets identity type.

Syntax: Here is the method signature:

```
public IdentityType getIdentityType()
```

getRoleAssignments

Gets role assignments for this identity.

Syntax: Here is the method signature:

```
public RoleAssignmentArray getRoleAssignments()
```

setAssociatedRoles

Sets the associated roles for this identity.

Syntax: Here is the method signature:

```
public void setAssociatedRoles(DNStringArray associatedRoles)
```

setEntityKey

Sets the identity entity key.

Syntax: Here is the method signature:

```
public void setEntityKey(java.lang.String entityKey)
```

setIdentityType

Sets the identity type.

Syntax: Here is the method signature:

```
public void setIdentityType(IdentityType identityType)
```

setRoleAssignments

Sets the role assignments for this identity.

Syntax: Here is the method signature:

```
public void setRoleAssignments(RoleAssignmentArray roleAssignments)
```

DNString

Class to hold a DN.

DNString constructors

The DNString class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
DNString()
```

Syntax 2: Here is the syntax for a constructor that takes a String as a parameter:

```
DNString(java.lang.String dn)
```

getDn

Gets the DN.

Syntax: Here is the method signature:

```
public java.lang.String getDn()
```

setDn

Sets the DN.

Syntax: Here is the method signature:

```
public void setDn(java.lang.String dn)
```

DNStringArray

This section provides reference information on the DNStringArray class.

DNStringArray constructors

The DNStringArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
DNStringArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of DNString objects as a parameter:

```
DNStringArray(DNString[] DNStringVal)
```

getDnstring

Returns an array of DNString objects.

Syntax: Here is the method signature:

```
DNString[] getDnstring()
```

setDnstring

Sets the array of DNString objects associated with the DNStringArray class.

Syntax: Here is the method signature:

```
void setDnstring(DNString[] DnstringVal)
```

Entitlement

Class to hold Entitlement information.

Entitlement constructors

The Entitlement class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
Entitlement()
```

getEntitlementDn

Gets the entitlement DN.

Syntax: Here is the method signature:

```
public java.lang.String getEntitlementDn()
```

getEntitlementParameters

Gets the entitlement parameters.

Syntax: Here is the method signature:

```
public java.lang.String getEntitlementParameters()
```

setEntitlementDn

Sets the entitlement DN.

Syntax: Here is the method signature:

```
public void setEntitlementDn(java.lang.String entitlementDn)
```

setEntitlementParameters

Sets the entitlement parameters.

Syntax: Here is the method signature:

```
public void setEntitlementParameters(java.lang.String
entitlementParameters)
```

EntitlementArray

This section provides reference information on the EntitlementArray class.

EntitlementArray constructors

The EntitlementArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
EntitlementArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of Entitlement objects as a parameter:

```
EntitlementArray(Entitlement[] EntitlementVal)
```

getEntitlement

Returns an array of Entitlement objects.

Syntax: Here is the method signature:

```
Entitlement[] getEntitlement()
```

setEntitlement

Sets the array of Entitlement objects associated with the EntitlementArray class.

Syntax: Here is the method signature:

```
void setEntitlement(EntitlementArray EntitlementVal)
```

Group

Class to represent a Group object.

Group constructors

The Group class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
Group()
```

getAssociatedRoles

Gets associated roles for this identity.

Syntax: Here is the method signature:

```
public DNStringArray getAssociatedRoles()
```

getDescription

Gets group description.

Syntax: Here is the method signature:

```
public java.lang.String getDescription()
```

getEntityKey

Gets identity entity key.

Syntax: Here is the method signature:

```
public java.lang.String getEntityKey()
```

getIdentityType

Gets identity type.

Syntax: Here is the method signature:

```
public IdentityType getIdentityType()
```

getRoleAssignments

Gets role assignments for this identity.

Syntax: Here is the method signature:

```
public RoleAssignmentArray getRoleAssignments()
```

setAssociatedRoles

Sets the associated roles for this identity.

Syntax: Here is the method signature:

```
public void setAssociatedRoles(DNStringArray associatedRoles)
```

setDescription

Sets the group description.

Syntax: Here is the method signature:

```
public void setDescription(java.lang.String description)
```

setEntityKey

Sets the identity entity key.

Syntax: Here is the method signature:

```
public void setEntityKey(java.lang.String entityKey)
```

setIdentityType

Sets the identity type.

Syntax: Here is the method signature:

```
public void setIdentityType(IdentityType identityType)
```

setRoleAssignments

Sets the role assignments for this identity.

Syntax: Here is the method signature:

```
public void setRoleAssignments(RoleAssignmentArray roleAssignments)
```

IdentityType

An JAX-RPC friendly representation of com.novell.idm.nrf.api.IdentityType.

Table 29-3 *Field summary*

Type	Name
static IdentityType	CONTAINER
static IdentityType	GROUP
static IdentityType	ROLE
static IdentityType	USER

IdentityType constructors

The IdentityType class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
IdentityType()
```

Syntax 2: Here is the syntax for a constructor that takes a String as a parameter:

```
IdentityType(java.lang.String value)
```

convertToAPI

Reconstructs an API representation object from an RPC representation.

Syntax: Here is the method signature:

```
public com.novell.idm.nrf.api.IdentityType convertToAPI()
```

convertToRPC

Contructs an RPC friendly representation from an API object.

Syntax: Here is the method signature:

```
public static IdentityType
convertToRPC(com.novell.idm.nrf.api.IdentityType type)
```

equals

This is an implementation of equals(). This implementation overrides the equals() method in java.lang.Object.

Syntax: Here is the method signature:

```
public boolean equals(java.lang.Object obj)
```

fromValue

This method is for WSSDK serialization.

Syntax: Here is the method signature:

```
public static IdentityType fromValue(java.lang.String value)
```

getValue

Gets the type.

Syntax: Here is the method signature:

```
public java.lang.String getValue()
```

hashCode

This is an implementation of hashCode(). This implementation overrides the hashCode() method in java.lang.Object.

Syntax: Here is the method signature:

```
public int hashCode()
```

setValue

Sets the type.

Syntax: Here is the method signature:

```
public void setValue(java.lang.String type)
```

toString

Implementation of toString() that returns a string representation of the class.

Syntax: Here is the method signature:

```
public java.lang.String toString()
```

IdentityTypeDnMap

Class to represent DNs grouped by identity type. Used for SOD violations.

IdentityTypeDnMap

The IdentityTypeDnMap class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
IdentityTypeDnMap()
```

Syntax 2: Here is the syntax for a constructor that takes a String as a parameter:

```
IdentityTypeDnMap(IdentityType identityType, DNStringArray dns)
```

getDns

Gets the DNs associated with the identity type.

Syntax: Here is the method signature:

```
public DNStringArray getDns()
```

getIdentityType

Gets identity type (USER, ROLE, GROUP, CONTAINER).

Syntax: Here is the method signature:

```
public IdentityType getIdentityType()
```

setDns

Sets the DNs to associate with the identity type.

Syntax: Here is the method signature:

```
public void setDns(DNStringArray dns)
```

setIdentityType

Sets the identity type (USER, ROLE, GROUP, or CONTAINER).

Syntax: Here is the method signature:

```
public void setIdentityType(IdentityType identityType)
```

IdentityTypeDnMapArray

This section provides reference information on the IdentityTypeDnMapArray class.

IdentityTypeDnMapArray constructors

The IdentityTypeDnMapArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
IdentityTypeDnMapArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of IdentityTypeDnMap objects as a parameter:

```
IdentityTypeDnMapArray(IdentityTypeDnMap[] IdentityTypeDnMapVal)
```

getIdentitytypednmap

Returns an array of IdentityTypeDnMap objects.

Syntax: Here is the method signature:

```
IdentityTypeDnMap[] getIdentitytypednmap()
```

setIdentitytypednmap

Sets the array of IdentityTypeDnMap objects associated with the IdentityTypeDnMapArray class.

Syntax: Here is the method signature:

```
void setIdentitytypednmap(IdentityTypeDnMap[] IdentityTypeDnMapVal)
```

LocalizedValue

The LocalizedValue class has been added to support management of localized strings for role definitions.

getValue

Returns a localized string value.

Syntax: Here is the method signature:

```
public String getValue()
```

setValue

Sets a localized string value.

Syntax: Here is the method signature:

```
public void setValue(final String value)
```

getLocale

Returns a string representaton of the Locale object.

Syntax: Here is the method signature:

```
public String getLocale()
```

setLocale

Sets a string representation of the Locale object.

Syntax: Here is the method signature:

```
public void setLocale()
```

LongArray

This section provides reference information on the LongArray class.

LongArray constructors

The LongArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
LongArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of Long objects as a parameter:

```
LongArray(long[] LongVal)
```

getLong

Returns an array of Long objects.

Syntax: Here is the method signature:

```
long[] getLong()
```

setLong

Sets the array of long objects associated with the LongArray class.

Syntax: Here is the method signature:

```
void setLong(LongArray LongVal)
```

NrfServiceException

This is the exception thrown by the remote Roles Web Service.

NrfServiceException constructors

The NrfServiceException class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
NrfServiceException()
```

Syntax 2: Here is the syntax for a constructor that takes a String as a parameter:

```
NrfServiceException(java.lang.String reason)
```

getReason

Returns the reason for the exception.

Syntax: Here is the method signature:

```
public java.lang.String getReason()
```

setReason

Sets the reason for the exception.

Syntax: Here is the method signature:

```
public void setReason(java.lang.String reason)
```

RequestCategoryType

An JAX-RPC friendly representation of com.novell.idm.nrf.persist.RequestCategoryType.

Table 29-4 *Field Summary*

Type	Name
static RequestCategoryType	ROLE_TO_CONTAINER_ADD
static RequestCategoryType	ROLE_TO_CONTAINER_ADD_SUBTREE
static RequestCategoryType	ROLE_TO_CONTAINER_REMOVE
static RequestCategoryType	ROLE_TO_GROUP_ADD
static RequestCategoryType	ROLE_TO_GROUP_REMOVE
static RequestCategoryType	ROLE_TO_ROLE_ADD
static RequestCategoryType	ROLE_TO_ROLE_REMOVE
static RequestCategoryType	ROLE_TO_USER_ADD
static RequestCategoryType	ROLE_TO_USER_REMOVE

RequestCategoryType constructors

The RequestCategoryType class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
RequestCategoryType()
```

Syntax 2: Here is the syntax for a constructor that takes a String as a parameter:

```
RequestCategoryType(java.lang.String value)
```

equals

Implementation of equals(). This implementation overrides the equals() method in java.lang.Object.

Syntax: Here is the method signature:

```
public boolean equals(java.lang.Object obj)
```

fromRPC

Reconstructs an API representation object from an RPC representation.

Syntax: Here is the method signature:

```
public com.novell.idm.nrf.persist.RequestCategoryType fromRPC() throws
com.novell.idm.nrf.exception.NrfException
```

fromValue

This method is for WSSDK serialization.

Syntax: Here is the method signature:

```
public static RequestCategoryType fromValue(java.lang.String value)
```

getValue

Gets the type.

Syntax: Here is the method signature:

```
public java.lang.String getValue()
```

hashCode

This implementation overrides the hashCode() method in java.lang.Object.

Syntax: Here is the method signature:

```
public int hashCode()
```

setValue

Sets the type.

Syntax: Here is the method signature:

```
public void setValue(java.lang.String type)
```

toRPC

Constructs an RPC friendly representation off of an API object.

Syntax: Here is the method signature:

```
public static RequestCategoryType
toRPC(com.novell.idm.nrf.persist.RequestCategoryType type)
```

toString

Implementation of toString() that returns a string representation of the class.

Syntax: Here is the method signature:

```
public java.lang.String toString()
```

RequestStatus

An JAX-RPC friendly representation of com.novell.idm.nrf.persist.RequestStatus.

Table 29-5 *Field Summary*

Type	Name
static RequestStatus	ACTIVATION_TIME_PENDING
static RequestStatus	APPROVAL_PENDING
static RequestStatus	APPROVAL_START_PENDING
static RequestStatus	APPROVAL_START_SUSPENDED
static RequestStatus	APPROVED
static RequestStatus	CLEANUP
static RequestStatus	DENIED
static RequestStatus	NEW_REQUEST
static RequestStatus	PROVISION
static RequestStatus	PROVISIONED
static RequestStatus	PROVISIONING_ERROR
static RequestStatus	SOD_APPROVAL_START_PENDING

Type	Name
static RequestStatus	SOD_APPROVAL_START_SUSPENDED
static RequestStatus	SOD_EXCEPTION_APPROVAL_PENDING
static RequestStatus	SOD_EXCEPTION_APPROVED
static RequestStatus	SOD_EXCEPTION_DENIED

RequestStatus constructors

The RequestStatus class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
RequestStatus()
```

Syntax 2: Here is the syntax for a constructor that takes a String as a parameter:

```
RequestStatus(java.lang.String value)
```

equals

Implementation of equals().

Syntax: Here is the method signature:

```
public boolean equals(java.lang.Object obj)
```

fromRPC

Reconstructs an API representation object from an RPC representation.

Syntax: Here is the method signature:

```
public com.novell.idm.nrf.persist.RequestStatus fromRPC() throws
com.novell.idm.nrf.exception.NrfException
```

fromValue

This method is for WSSDK serialization.

Syntax: Here is the method signature:

```
public static RequestStatus fromValue(java.lang.String value)
```

getValue

Gets the type.

Syntax: Here is the method signature:

```
public java.lang.String getValue()
```

hashCode

This implementation overrides the hashCode() method in java.lang.Object.

Syntax: Here is the method signature:

```
public int hashCode()
```

setValue

Sets the type.

Syntax: Here is the method signature:

```
public void setValue(java.lang.String type)
```

toRPC

Constructs an RPC friendly representation off of an API object.

Syntax: Here is the method signature:

```
public static RequestStatus toRPC(com.novell.idm.nrf.persist.RequestStatus
type)
```

toString

Implementation of toString() that returns a string representation of the class.

Syntax: Here is the method signature:

```
public java.lang.String toString()
```

ResourceAssociation

Supporting class that holds information about resource associations for a role.

getRole

Returns the DN for the role involved in the association.

```
public String getRole()
```

setRole

Sets the DN for the role involved in the association.

```
public void setRole(String role)
```

getEntityKey

Returns the entity key for the association.

```
public String getEntityKey()
```

setEntityKey

Sets the entity key for the association.

```
public void setEntityKey(String entityKey)
```

getResource

Returns the DN for the resource involved in the association.

```
public String getResource()
```

setResource

Sets the DN for the resource involved in the association.

```
public void setResource(String resource)
```

getDynamicParameters

Returns the list of dynamic parameters for the resource.

```
public DynamicParameter[] getDynamicParameters()
```

setDynamicParameters

Sets the list of dynamic parameters for the resource.

```
public void setDynamicParameters(DynamicParameter[] parameterValues)
```

getLocalizedDescriptions

Returns the list of localized descriptions.

```
public LocalizedValue[] getLocalizedDescriptions()
```

setLocalizedDescriptions

Sets the list of localized descriptions.

```
public void setLocalizedDescriptions(LocalizedValue[] descriptions)
```

getApprovalOverride

Returns the boolean flag indicating whether the role approval process overrides the resource approval process.

```
public boolean getApprovalOverride()
```

setApprovalOverride

Sets the boolean flag indicating whether the role approval process overrides the resource approval process.

```
public void setApprovalOverride(boolean override)
```

getStatus

Returns the status of the association.

```
public int getStatus()
```

setStatus

Sets the status of the association.

```
public void setStatus(int status)
```

toString

Converts the resource association to a string.

```
public String toString()
```

Role

Value class to hold the role information.

Role constructors

The Role class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
Role()
```

getApprovers

Gets the approvers of the role approval.

Syntax: Here is the method signature:

```
public ApproverArray getApprovers()
```

getAssociatedRoles

Gets the associated roles.

Syntax: Here is the method signature:

```
public DNStringArray getAssociatedRoles()
```

getChildRoles

Gets the children roles.

Syntax: Here is the method signature:

```
public DNStringArray getChildRoles()
```

getDescription

Gets the role description.

Syntax: Here is the method signature:

```
public java.lang.String getDescription()
```

getEntitlementRef

Gets the entitlement references.

Syntax: Here is the method signature:

```
public EntitlementArray getEntitlementRef()
```

getEntityKey

Gets the role entity key.

Syntax: Here is the method signature:

```
public java.lang.String getEntityKey()
```

getImplicitContainers

Gets the implicit container DNs.

Syntax: Here is the method signature:

```
public DNStringArray getImplicitContainers()
```

getImplicitGroups

Gets implicit group DNs.

Syntax: Here is the method signature:

```
public DNStringArray getImplicitGroups()
```

getName

Gets the role name.

Syntax: Here is the method signature:

```
public java.lang.String getName()
```

getOwners

Gets the owner DNs.

Syntax: Here is the method signature:

```
public DNStringArray getOwners()
```

getParentRoles

Gets the parent roles.

Syntax: Here is the method signature:

```
public DNStringArray getParentRoles()
```

getQuorum

Gets the quorum amount.

Syntax: Here is the method signature:

```
public java.lang.String getQuorum()
```

getRequestDef

Gets the request definition for approval processing.

Syntax: Here is the method signature:

```
public java.lang.String getRequestDef()
```

getRoleAssignments

Gets the role assignments.

Syntax: Here is the method signature:

```
public RoleAssignmentArray getRoleAssignments()
```

getRoleCategoryKeys

Gets the role category keys.

Syntax: Here is the method signature:

```
public CategoryKeyArray getRoleCategoryKeys()
```

getRoleLevel

Gets the role level object.

Syntax: Here is the method signature:

```
public RoleLevel getRoleLevel()
```

getSystemRole

Gets the system role flag.

Syntax: Here is the method signature:

```
public boolean getSystemRole()
```

setApprovers

Sets the approvers for role approval processing.

Syntax: Here is the method signature:

```
public void setApprovers(ApproverArray approvers)
```

setAssociatedRoles

Sets the associated roles.

Syntax: Here is the method signature:

```
public void setAssociatedRoles(DNStringArray associatedRoles)
```

setChildRoles

Sets the children roles.

Syntax: Here is the method signature:

```
public void setChildRoles(DNStringArray childRoles)
```

setDescription

Sets the role description.

Syntax: Here is the method signature:

```
public void setDescription(java.lang.String description)
```

setEntitlementRef

Sets the entitlement references.

Syntax: Here is the method signature:

```
public void setEntitlementRef(EntitlementArray entitlementRef)
```

setEntityKey

Sets the role entity key.

Syntax: Here is the method signature:

```
public void setEntityKey(java.lang.String entityKey)
```

setImplicitContainers

Sets the implicit container DNs.

Syntax: Here is the method signature:

```
public void setImplicitContainers(DNStringArray implicitContainers)
```

setImplicitGroups

Sets the implicit group DNs.

Syntax: Here is the method signature:

```
public void setImplicitGroups(DNStringArray implicitGroups)
```

setName

Sets the role name.

Syntax: Here is the method signature:

```
public void setName(java.lang.String name)
```

setOwners

Sets the owner DNs.

Syntax: Here is the method signature:

```
public void setOwners(DNStringArray owners)
```

setParentRoles

Sets the parent roles.

Syntax: Here is the method signature:

```
public void setParentRoles(DNStringArray parentRoles)
```

setQuorum

Sets the quorum amount.

Syntax: Here is the method signature:

```
public void setQuorum(java.lang.String quorum)
```

setRequestDef

Sets the request definition for approval processing.

Syntax: Here is the method signature:

```
public void setRequestDef(java.lang.String requestDef)
```

setRoleAssignments

Sets the role assignments.

Syntax: Here is the method signature:

```
public void setRoleAssignments(RoleAssignmentArray roleAssignments)
```

setRoleCategoryKeys

Sets the role category keys.

Syntax: Here is the method signature:

```
public void setRoleCategoryKeys(CategoryKeyArray roleCategoryKeys)
```

setRoleLevel

Sets the role level object.

Syntax: Here is the method signature:

```
public void setRoleLevel(RoleLevel roleLevel)
```

setSystemRole

Sets the system role flag.

Syntax: Here is the method signature:

```
public void setSystemRole(boolean systemRole)
```

RoleAssignment

Value class to hold role assignment information.

RoleAssignment

The RoleAssignment class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
RoleAssignment()
```

getAssignmentType

Gets the role assignment type.

Syntax: Here is the method signature:

```
public RoleAssignmentType getAssignmentType()
```

getCauseIdentities

Gets the cause identities DNs.

Syntax: Here is the method signature:

```
public IdentityTypeDnMapArray getCauseIdentities()
```

getEffectiveDate

Gets the effective date.

Syntax: Here is the method signature:

```
public java.util.Date getEffectiveDate()
```

getExpirationDate

Gets the expiration date.

Syntax: Here is the method signature:

```
public java.util.Date getExpirationDate()
```

getExplicitIdentities

Gets the explicit identities DNs.

Syntax: Here is the method signature:

```
public DNStringArray getExplicitIdentities()
```

getRole

Gets the role associated with the assignment.

Syntax: Here is the method signature:

```
public java.lang.String getRole()
```

setAssignmentType

Sets the role assignment type.

Syntax: Here is the method signature:

```
public void setAssignmentType(RoleAssignmentType assignmentType)
```

setCauseIdentities

Sets the cause identities DNs.

Syntax: Here is the method signature:

```
public void setCauseIdentities(IdentityTypeDnMapArray causeIdentities)
```

setEffectiveDate

Sets the effective date.

Syntax: Here is the method signature:

```
public void setEffectiveDate(java.util.Date effectiveDate)
```

setExpirationDate

Sets the expiration date.

Syntax: Here is the method signature:

```
public void setExpirationDate(java.util.Date expirationDate)
```

setExplicitIdentities

Sets the explicit identities DNs.

Syntax: Here is the method signature:

```
public void setExplicitIdentities(DNStringArray explicitIdentities)
```

setRole

Sets role associated with this assignment.

Syntax: Here is the method signature:

```
public void setRole(java.lang.String role)
```

RoleAssignmentArray

This section provides reference information on the RoleAssignmentArray class.

RoleAssignmentArray constructors

The RoleAssignmentArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
RoleAssignmentArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of Attribute objects as a parameter:

```
RoleAssignmentArray(RoleAssignment[] RoleAssignmentVal)
```

getRoleassignment

Returns an array of RoleAssignment objects.

Syntax: Here is the method signature:

```
RoleAssignment[] getRoleassignment()
```

setRoleassignment

Sets the array of RoleAssignment objects associated with the RoleAssignmentArray class.

Syntax: Here is the method signature:

```
void setRoleassignment (RoleAssignment[] RoleAssignmentVal)
```

RoleAssignmentActionType

An JAX-RPC friendly representation of com.novell.idm.nrf.RoleAssignmentActionType.

Table 29-6 *Field Summary*

Type	Name
static RoleAssignmentActionType	EXTEND
static RoleAssignmentActionType	GRANT
static RoleAssignmentActionType	REVOKE

RoleAssignmentActionType constructors

The RoleAssignmentActionType class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
RoleAssignmentActionType()
```

Syntax 2: Here is the syntax for a constructor that takes a String as a parameter:

```
RoleAssignmentActionType(java.lang.String value)
```

equals

Implementation of equals().

Syntax: Here is the method signature:

```
public boolean equals(java.lang.Object obj)
```

fromRPC

Reconstructs an API representation object from an RPC representation.

Syntax: Here is the method signature:

```
public com.novell.idm.nrf.RoleAssignmentActionType fromRPC()
```

fromValue

This method is for WSSDK serialization.

Syntax: Here is the method signature:

```
public static RoleAssignmentActionType fromValue(java.lang.String value)
```

getValue

Gets the type.

Syntax: Here is the method signature:

```
public java.lang.String getValue()
```

hashCode

This is an implementation of hashCode(). This implementation overrides the hashCode() method in java.lang.Object.

Syntax: Here is the method signature:

```
public int hashCode()
```

setValue

Sets the type.

Syntax: Here is the method signature:

```
public void setValue(java.lang.String type)
```

toRPC

Constructs an RPC friendly representation off of an API object.

Syntax: Here is the method signature:

```
public static RoleAssignmentActionType
toRPC(com.novell.idm.nrf.RoleAssignmentActionType type)
```

toString

Implementation of toString() that returns a string representation of the class.

Syntax: Here is the method signature:

```
public java.lang.String toString()
```

RoleAssignmentRequest

Class to represent a role assignment request.

RoleAssignmentRequest

The RoleAssignmentRequest class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
RoleAssignmentRequest()
```

getActionType

Gets role assignment type (grant, revoke, extend).

Syntax: Here is the method signature:

```
public RoleAssignmentActionType getActionType()
```

getAssignmentType

Gets the role assignment type.

Syntax: Here is the method signature:

```
public RoleAssignmentType getAssignmentType()
```

getCorrelationID

Gets the correlation ID.

Syntax: Here is the method signature:

```
public java.lang.String getCorrelationID()
```

getEffectiveDate

Gets the effective date.

Syntax: Here is the method signature:

```
public java.util.Date getEffectiveDate()
```

getExpirationDate

Gets the expiration date.

Syntax: Here is the method signature:

```
public java.util.Date getExpirationDate()
```

getIdentity

Gets the identity to assign roles to.

Syntax: Here is the method signature:

```
public java.lang.String getIdentity()
```

getReason

Gets the reason for the role assignment.

Syntax: Here is the method signature:

```
public java.lang.String getReason()
```

getRoles

Gets the roles to assign to the identity.

Syntax: Here is the method signature:

```
public DNStringArray getRoles()
```

getSodOveridesRequested

Gets the SOD DNs and justification to override.

Syntax: Here is the method signature:

```
public SodJustificationArray getSodOveridesRequested()
```

setActionType

Sets the action type (grant, revoke, extend).

Syntax: Here is the method signature:

```
public void setActionType(RoleAssignmentActionType actionType)
```

setAssignmentType

Sets the role assignment type.

Syntax: Here is the method signature:

```
public void setAssignmentType(RoleAssignmentType assignmentType)
```

setCorrelationID

Sets the correlation ID.

Syntax: Here is the method signature:

```
public void setCorrelationID(java.lang.String correlationID)
```

setEffectiveDate

Sets the effective date.

Syntax: Here is the method signature:

```
public void setEffectiveDate(java.util.Date effectiveDate)
```

setExpirationDate

Sets the expiration date.

Syntax: Here is the method signature:

```
public void setExpirationDate(java.util.Date expirationDate)
```

setIdentity

Sets the identity to assign roles to.

Syntax: Here is the method signature:

```
public void setIdentity(java.lang.String identity)
```

setReason

Sets the reason for the role assignment.

Syntax: Here is the method signature:

```
public void setReason(java.lang.String reason)
```

setRoles

Sets the roles to assign to the identity.

Syntax: Here is the method signature:

```
public void setRoles(DNStringArray roles)
```

setSodOveridesRequested

Sets the SOD DNs and justification to override.

Syntax: Here is the method signature:

```
public void setSodOveridesRequested(SodJustificationArray
sodOveridesRequested)
```

RoleAssignmentRequestStatus

This class represents the status of a role assignment.

RoleAssignmentRequestStatus

The RoleAssignmentRequestStatus class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
RoleAssignmentRequestStatus()
```

getCategory

Gets the request category.

Syntax: Here is the method signature:

```
public RequestCategoryType getCategory()
```

getCorrelationId

Gets the correlation ID.

Syntax: Here is the method signature:

```
public java.lang.String getCorrelationId()
```

getEffectiveDate

Gets the effective date.

Syntax: Here is the method signature:

```
public java.util.Date getEffectiveDate()
```

getEntityKey

Gets the entity key.

Syntax: Here is the method signature:

```
public java.lang.String getEntityKey()
```

getExpirationDate

Gets the expiration date.

Syntax: Here is the method signature:

```
public java.util.Date getExpirationDate()
```

getReason

Gets the reason for the role assignment.

Syntax: Here is the method signature:

```
public java.lang.String getReason()
```

getRequestDate

Gets the request date.

Syntax: Here is the method signature:

```
public java.util.Date getRequestDate()
```

getRequester

Gets the request DN.

Syntax: Here is the method signature:

```
public java.lang.String getRequester()
```

getSource

Gets the source Role DN.

Syntax: Here is the method signature:

```
public java.lang.String getSource()
```

getStatus

Gets the request status.

Syntax: Here is the method signature:

```
public RequestStatus getStatus()
```

getTarget

Gets the targeted identity DN.

Syntax: Here is the method signature:

```
public java.lang.String getTarget()
```

setCategory

Sets the request category.

Syntax: Here is the method signature:

```
public void setCategory(RequestCategoryType category)
```

setCorrelationId

Sets the correlation ID.

Syntax: Here is the method signature:

```
public void setCorrelationId(java.lang.String correlationId)
```

setEffectiveDate

Sets the effective date.

Syntax: Here is the method signature:

```
public void setEffectiveDate(java.util.Date effectiveDate)
```

setEntityKey

Sets the entity key.

Syntax: Here is the method signature:

```
public void setEntityKey(java.lang.String entityKey)
```

setExpirationDate

Sets the expiration date.

Syntax: Here is the method signature:

```
public void setExpirationDate(java.util.Date expirationDate)
```

setReason

Sets the reason for the role assignment.

Syntax: Here is the method signature:

```
public void setReason(java.lang.String reason)
```

setRequestDate

Sets the request date.

Syntax: Here is the method signature:

```
public void setRequestDate(java.util.Date requestDate)
```

setRequester

Sets the requester DN.

Syntax: Here is the method signature:

```
public void setRequester(java.lang.String requester)
```

setSource

Sets the source Role DN.

Syntax: Here is the method signature:

```
public void setSource(java.lang.String source)
```

setStatus

Sets the request status.

Syntax: Here is the method signature:

```
public void setStatus(RequestStatus status)
```

setTarget

Sets the identity targeted DN.

Syntax: Here is the method signature:

```
public void setTarget(java.lang.String target)
```

RoleAssignmentType

An JAX-RPC friendly representation of com.novell.idm.nrf.RoleAssignmentType.

Table 29-7 *Field Summary*

Type	Name
static RoleAssignmentType	CONTAINER_TO_ROLE
static RoleAssignmentType	CONTAINER_WITH_SUBTREE_TO_ROLE
static RoleAssignmentType	GROUP_TO_ROLE
static RoleAssignmentType	ROLE_TO_ROLE
static RoleAssignmentType	USER_TO_ROLE

RoleAssignmentType constructors

The CategoryKey class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
CategoryKey()
```

Syntax 2: Here is the syntax for a constructor that takes a String as a parameter:

```
CategoryKey(java.lang.String categoryKey)
```

convertToAPI

Reconstructs an API representation object from an RPC representation.

Syntax: Here is the method signature:

```
public com.novell.idm.nrf.RoleAssignmentType convertToAPI()
```

convertToRPC

Constructs an RPC friendly representation off of an API object.

Syntax: Here is the method signature:

```
public static RoleAssignmentType
convertToRPC(com.novell.idm.nrf.RoleAssignmentType type)
```

equals

Implementation of equals().

Syntax: Here is the method signature:

```
public boolean equals(java.lang.Object obj)
```

fromValue

This method is for WSSDK serialization.

Syntax: Here is the method signature:

```
public static RoleAssignmentType fromValue(java.lang.String value)
```

getValue

Gets the type.

Syntax: Here is the method signature:

```
public java.lang.String getValue()
```

hashCode

This is an implementation of hashCode(). This implementation overrides the hashCode() method in java.lang.Object.

Syntax: Here is the method signature:

```
public int hashCode()
```

setValue

Sets the type.

Syntax: Here is the method signature:

```
public void setValue(java.lang.String type)
```

toString

Implementation of toString() that returns a string representation of the class.

Syntax: Here is the method signature:

```
public java.lang.String toString()
```

RoleAssignmentTypeInfo

An JAX-RPC friendly representation of the details of the com.novell.idm.nrf.RoleAssignmentType enumeration.

RoleAssignmentTypeInfo

The RoleAssignmentTypeInfo class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
RoleAssignmentTypeInfo()
```

convertToRPC

Constructs an RPC friendly representation from an API object.

Syntax: Here is the method signature:

```
public static RoleAssignmentTypeInfo
convertToRPC(com.novell.idm.nrf.RoleAssignmentType type)
```

getIdentityType

Returns the JAX-RPC friendly identity type.

Syntax: Here is the method signature:

```
public IdentityType getIdentityType()
```

getSubtreeIncluded

Determines whether the sub tree is included.

Syntax: Here is the method signature:

```
public boolean getSubtreeIncluded()
```

getSupportsApproval

Determines whether the assignment supports approval.

Syntax: Here is the method signature:

```
public boolean getSupportsApproval()
```

getSupportsEffectiveDate

Determines whether the assignment supports an effective date.

Syntax: Here is the method signature:

```
public boolean getSupportsEffectiveDate()
```

getSupportsExpiration

Determines whether the assignment supports expiration.

Syntax: Here is the method signature:

```
public boolean getSupportsExpiration()
```

getSupportsSODApproval

Determines whether the assignment supports SOD approval.

Syntax: Here is the method signature:

```
public boolean getSupportsSODApproval()
```

setIdentityType

Sets the JAX-RPC friendly identity type.

Syntax: Here is the method signature:

```
public void setIdentityType(IdentityType type)
```

setSubtreeIncluded

Sets whether the sub tree is included.

Syntax: Here is the method signature:

```
public void setSubtreeIncluded(boolean bool)
```

setSupportsApproval

Sets whether the assignment supports approval.

Syntax: Here is the method signature:

```
public void setSupportsApproval(boolean bool)
```

setSupportsEffectiveDate

Sets whether the assignment supports effective date.

Syntax: Here is the method signature:

```
public void setSupportsEffectiveDate(boolean bool)
```

setSupportsExpiration

Sets whethers the assignment supports expiration.

Syntax: Here is the method signature:

```
public void setSupportsExpiration(boolean bool)
```

setSupportsSODApproval

Sets whether the assignment supports SOD approval.

Syntax: Here is the method signature:

```
public void setSupportsSODApproval(boolean bool)
```

RoleInfo

Value class to hold main role information. This is a small subset of the role value class.

RoleInfo constructors

The RoleInfo class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
RoleInfo()
```

getDescription

Gets the role description.

Syntax: Here is the method signature:

```
public java.lang.String getDescription()
```

getEntityKey

Gets the role entity key.

Syntax: Here is the method signature:

```
public java.lang.String getEntityKey()
```

getName

Gets the role name.

Syntax: Here is the method signature:

```
public java.lang.String getName()
```

getRoleCategoryKeys

Gets the role category keys.

Syntax: Here is the method signature:

```
public CategoryKeyArray getRoleCategoryKeys()
```

getRoleLevel

Gets the role level object.

Syntax: Here is the method signature:

```
public RoleLevel getRoleLevel()
```

setDescription

Sets the role description.

Syntax: Here is the method signature:

```
public void setDescription(java.lang.String description)
```

setEntityKey

Sets the role entity key.

Syntax: Here is the method signature:

```
public void setEntityKey(java.lang.String entityKey)
```

setName

Sets the role name.

Syntax: Here is the method signature:

```
public void setName(java.lang.String name)
```

setRoleCategoryKeys

Sets the role category keys.

Syntax: Here is the method signature:

```
public void setRoleCategoryKeys(CategoryKeyArray roleCategoryKeys)
```

setRoleLevel

Sets role level object.

Syntax: Here is the method signature:

```
public void setRoleLevel(RoleLevel roleLevel)
```

RoleInfoArray

This section provides reference information on the RoleInfoArray class.

RoleInfoArray constructors

The RoleInfoArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
RoleInfoArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of Attribute objects as a parameter:

```
RoleInfoArray(RoleInfo[] RoleInfoVal)
```

getRoleinfo

Returns an array of RoleInfo objects.

Syntax: Here is the method signature:

```
RoleInfo[] getRoleinfo()
```

setRoleinfo

Sets the array of RoleInfo objects associated with the RoleInfoArray class.

Syntax: Here is the method signature:

```
void setRoleinfo (RoleInfo[] RoleInfoVal)
```

RoleLevel

This class represent a role level.

RoleLevel constructors

The RoleLevel class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
RoleLevel()
```

getContainer

Gets the role level container.

Syntax: Here is the method signature:

```
public java.lang.String getContainer()
```

getDescription

Gets the role level description.

Syntax: Here is the method signature:

```
public java.lang.String getDescription()
```

getLevel

Gets the role level.

Syntax: Here is the method signature:

```
public long getLevel()
```

getName

Gets the role level name.

Syntax: Here is the method signature:

```
public java.lang.String getName()
```

setContainer

Sets the role level container.

Syntax: Here is the method signature:

```
public void setContainer(java.lang.String container)
```

setDescription

Sets the role level description.

Syntax: Here is the method signature:

```
public void setDescription(java.lang.String description)
```

setLevel

Sets the role level.

Syntax: Here is the method signature:

```
public void setLevel(long level)
```

setName

Sets the role level name.

Syntax: Here is the method signature:

```
public void setName(java.lang.String name)
```

RoleLevelArray

This section provides reference information on the RoleLevelArray class.

RoleLevelArray constructors

The RoleLevelArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
RoleLevelArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of Attribute objects as a parameter:

```
RoleLevelArray(RoleLevel[] RoleLevelVal)
```

getRolelevel

Returns an array of RoleLevel objects.

Syntax: Here is the method signature:

```
RoleLevel[] getRolelevel()
```

setRolelevel

Sets the array of RoleLevel objects associated with the RoleLevelArray class.

Syntax: Here is the method signature:

```
void setRolelevel (RoleLevel[] RoleLevelVal)
```

RoleRequest

The Role Request class has been added to support the creation of roles. The Role Request class is a value class used to hold information about a request to create a role.

getName

Gets the role name.

Syntax: Here is the method signature:

```
public String getName()
```

getDescription

Gets the role description.

Syntax: Here is the method signature:

```
public String getDescription()
```

getEntityKey

Gets the entity key for the role.

Syntax: Here is the method signature:

```
public String getEntityKey()
```

getRoleLevel

Gets the role level object.

Syntax: Here is the method signature:

```
public long getRoleLevel()
```

getRoleCategoryKeys

Gets the role category keys.

Syntax: Here is the method signature:

```
public CategoryKey[] getRoleCategoryKeys()
```

getQuorum

Gets the quorum amount.

Syntax: Here is the method signature:

```
public String getQuorum()
```

getRequestDef

Gets the provisioning request definition for approval processing.

Syntax: Here is the method signature:

```
public String getRequestDef()
```

getApprovers

Gets the approvers for the role definition.

Syntax: Here is the method signature:

```
public Approver[] getApprovers()
```

getOwners

Gets the owner DNs.

Syntax: Here is the method signature:

```
public DNString[] getOwners()
```

getRoleAssignments

Gets the associated roles.

Syntax: Here is the method signature:

```
public String getRoleAssignments()
```

getSystemRole

Gets the system role flag, which indicates whether this is a system role.

Syntax: Here is the method signature:

```
public boolean getSystemRole()
```

getContainer

Gets the name of the role container.

Syntax: Here is the method signature:

```
public String getContainer()
```

setName

Sets the role name.

Syntax: Here is the method signature:

```
public void setName()
```

setDescription

Sets the role description.

Syntax: Here is the method signature:

```
public void setDescription()
```

setEntityKey

Sets the entity key for the role.

Syntax: Here is the method signature:

```
public void setEntityKey()
```

setRoleLevel

Sets the role level object.

Syntax: Here is the method signature:

```
public void setRoleLevel()
```

setRoleCategoryKeys

Sets the role category keys.

Syntax: Here is the method signature:

```
public void setRoleCategoryKeys()
```

setQuorum

Sets the quorum amount.

Syntax: Here is the method signature:

```
public void setQuorum()
```

setRequestDef

Sets the provisioning request definition for approval processing.

Syntax: Here is the method signature:

```
public void setRequestDef()
```

setApprovers

Sets the approvers for role approval processing.

Syntax: Here is the method signature:

```
public void setApprovers()
```

setOwners

Sets the owner DNs.

Syntax: Here is the method signature:

```
public void setOwners()
```

setSystemRole

Sets the system role flag, which determines whether this is a system role.

Syntax: Here is the method signature:

```
public void setSystemRole()
```

setContainer

Sets the role container.

Syntax: Here is the method signature:

```
public void setContainer()
```

RoleServiceDelegate

Delegate class to perform the actual call to the API layer. Should be used by all skeleton classes.

RoleServiceDelegate constructors

The RoleServiceDelegate class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
RoleServiceDelegate(com.novell.srvprv.spi.security.ISecurityContext ctx,
java.util.Locale locale)
```

findSodByExample

Finds all SoD objects based on the search criteria in the given SOD object.

Syntax: Here is the method signature:

```
SodArray findSodByExample(Sod sod) throws NrfServiceException,
java.rmi.RemoteException
```

findSodByExampleWithOperator

Finds all SoD objects based on the search criteria found in the given SOD object

Syntax: Here is the method signature:

```
SodArray findSodByExampleWithOperator(Sod searchCriteria, boolean
useAndForMultiValueSearch) throws NrfServiceException,
java.rmi.RemoteException
```

findSodById

Find by key.

Syntax: Here is the method signature:

```
Sod findSodById(java.lang.String entityKey) throws NrfServiceException,
java.rmi.RemoteException
```

getAssignedIdentities

Returns a list of role assignments for a specified identity.

Syntax: Here is the method signature:

```
RoleAssignmentArray   getAssignedIdentities(java.lang.String identityDn,
IdentityType type, boolean direct) throws NrfServiceException,
java.rmi.RemoteException
```

getConfiguration

Returns the role system configuration defined in the role vault root (nrfConfiguration)

Syntax: Here is the method signature:

```
Configuration getConfiguration() throws NrfServiceException,
java.rmi.RemoteException
```

getContainer

Gets container and role information for a given container DN.

Syntax: Here is the method signature:

```
Container getContainer(java.lang.String containerDn)
throws NrfServiceException, java.rmi.RemoteException
```

getExceptionList

Returns a list of Sod instances for all SOD violations found for a specific identity and type.

Syntax: Here is the method signature:

```
SodArray getExceptionsList(java.lang.String identity, IdentityType
identityType) throws NrfServiceException, java.rmi.RemoteException
```

getGroup

Gets group and role information for a given group DN.

Syntax: Here is the method signature:

```
Group getGroup(java.lang.String groupDn) throws NrfServiceException,
java.rmi.RemoteException
```

getIdentitiesInViolation

Returns a map of identities which are in violation of a given SoD.

Syntax: Here is the method signature:

```
IdentityTypeDnMapArray getIdentitiesInViolation(java.lang.String sodDn)
throws NrfServiceException, java.rmi.RemoteException
```

getIdentityRoleConflicts

Returns a list of Sod instances for all SOD conflicts found for a given list of roles for a given identity.

Syntax: Here is the method signature:

```
SodArray getIdentityRoleConflicts(java.lang.String identity, IdentityType
identityType, DNStringArray requestedRoles) throws NrfServiceException,
java.rmi.RemoteException
```

getRole

Retrieves a role object defined by a role DN

Syntax: Here is the method signature:

```
Role getRole(java.lang.String roleDn) throws NrfServiceException,
java.rmi.RemoteException
```

getRoleAssignmentRequestStatus

Returns a list of role assignment request status instances given a correlation ID.

Syntax: Here is the method signature:

```
RoleAssignmentRequestStatusArray
getRoleAssignmentRequestStatus(java.lang.String correlationId) throws
NrfServiceException, java.rmi.RemoteException
```

getRoleAssignmentRequestStatusByIdentityType

Returns a list of role assignment request status instances given an identity and an identity type.

Syntax: Here is the method signature:

```
RoleAssignmentRequestStatusArray
getRoleAssignmentRequestStatusByIdentityType(java.lang.String identityDn,
IdentityType identityType) throws NrfServiceException,
java.rmi.RemoteException
```

getRoleAssignmentTypeInfo

Retrieves details about a RoleAssignmentType.

Syntax: Here is the method signature:

```
RoleAssignmentTypeInfo getRoleAssignmentTypeInfo(RoleAssignmentType type)
throws NrfServiceException, java.rmi.RemoteException
```

getRoleCategories

Gets role categories.

Syntax: Here is the method signature:

```
CategoryArray getRoleCategories() throws NrfServiceException,
java.rmi.RemoteException
```

getRoleConflicts

Returns a list of Sod instances found for all given roles. This method always returns a list.

Syntax: Here is the method signature:

```
SodArray getRoleConflicts(DNStringArray roles) throws NrfServiceException,
java.rmi.RemoteException
```

getRoleLevels

Gets role levels.

Syntax: Here is the method signature:

```
RoleLevelArray getRoleLevels() throws NrfServiceException,
java.rmi.RemoteException
```

getRolesInfo

Returns a list of RoleInfo instances given a list of role DNs.

Syntax: Here is the method signature:

```
RoleInfoArray getRolesInfo(DNStringArray roleDns) throws
NrfServiceException, java.rmi.RemoteException
```

getRolesInfoByCategory

Returns a list of RoleInfo instances given a list of role category keys.

Syntax: Here is the method signature:

```
RoleInfoArray getRolesInfoByCategory(CategoryKeyArray roleCategoryKeys)
throws NrfServiceException, java.rmi.RemoteException
```

getRolesInfoByLevel

Returns a list of RoleInfo instances given a list of role levels.

Syntax: Here is the method signature:

```
RoleInfoArray getRolesInfoByLevel(LongArray roleLevels) throws
NrfServiceException, java.rmi.RemoteException
```

getTargetSourceConflicts

Returns a list of Sod instances for all SOD conflicts defined between the target role DN and the source role DN.

Syntax: Here is the method signature:

```
SodArray getTargetSourceConflicts(java.lang.String targetName,
java.lang.String sourceName) throws NrfServiceException,
java.rmi.RemoteException
```

getUser

Gets user info including all role assignments for a given user DN stored in a UserIdentity object.

Syntax: Here is the method signature:

```
User getUser(java.lang.String userDn) throws NrfServiceException,
java.rmi.RemoteException
```

getVersion

Returns the version of this Web Service.

Syntax: Here is the method signature:

```
VersionVO getVersion() throws java.rmi.RemoteException
```

isUserInRole

Returns boolean flag; true if role has been assigned to a User identity

Syntax: Here is the method signature:

```
boolean isUserInRole(java.lang.String userDn, java.lang.String roleDn)
```

requestRoleAssignment

Returns a list of request DNs created by the role assignment. Be aware that the role assignment expires only if the role is assigned to a user and not when it is assigned to a group or a container.

Syntax: Here is the method signature:

```
DNStringArray requestRolesAssignment(RoleAssignmentRequest
roleAssignmentRequest) throws NrfServiceException,
java.rmi.RemoteException
```

RoleServiceSkeletonImpl

Class to represent the skeleton server side implementation of the Role Based offered services.

RoleServiceSkeletonImpl

The RoleServiceSkeletonImpl class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
RoleServiceSkeletonImpl()
```

findSodByExample

Finds all SoD objects based on the search criteria in the given SOD object.

Syntax: Here is the method signature:

```
SodArray findSodByExample(Sod sod) throws NrfServiceException,
java.rmi.RemoteException
```

findSodByExampleWithOperator

Finds all SoD objects based on the search criteria found in the given SOD object

Syntax: Here is the method signature:

```
SodArray findSodByExampleWithOperator(Sod searchCriteria, boolean
useAndForMultiValueSearch) throws NrfServiceException,
java.rmi.RemoteException
```

findSodById

Find by key.

Syntax: Here is the method signature:

```
Sod findSodById(java.lang.String entityKey) throws NrfServiceException,
java.rmi.RemoteException
```

getAssignedIdentities

Returns a list of role assignments for a specified identity.

Syntax: Here is the method signature:

```
RoleAssignmentArray   getAssignedIdentities(java.lang.String identityDn,
IdentityType type, boolean direct) throws NrfServiceException,
java.rmi.RemoteException
```

getConfiguration

Returns the role system configuration defined in the role vault root (nrfConfiguration)

Syntax: Here is the method signature:

```
Configuration getConfiguration() throws NrfServiceException,
java.rmi.RemoteException
```

getContainer

Gets container and role information for a given container DN.

Syntax: Here is the method signature:

```
Container getContainer(java.lang.String containerDn)
throws NrfServiceException, java.rmi.RemoteException
```

getExceptionList

Returns a list of Sod instances for all SOD violations found for a specific identity and type.

Syntax: Here is the method signature:

```
SodArray getExceptionsList(java.lang.String identity, IdentityType
identityType) throws NrfServiceException, java.rmi.RemoteException
```

getGroup

Gets group and role information for a given group DN.

Syntax: Here is the method signature:

```
Group getGroup(java.lang.String groupDn) throws NrfServiceException,
java.rmi.RemoteException
```

getIdentitiesInViolation

Returns a map of identities which are in violation of a given SoD.

Syntax: Here is the method signature:

```
IdentityTypeDnMapArray getIdentitiesInViolation(java.lang.String sodDn)
throws NrfServiceException, java.rmi.RemoteException
```

getIdentityRoleConflicts

Returns a list of Sod instances for all SOD conflicts found for a given list of roles for a given identity.

Syntax: Here is the method signature:

```
SodArray getIdentityRoleConflicts(java.lang.String identity, IdentityType
identityType, DNStringArray requestedRoles) throws NrfServiceException,
java.rmi.RemoteException
```

getRole

Retrieves a role object defined by a role DN

Syntax: Here is the method signature:

```
Role getRole(java.lang.String roleDn) throws NrfServiceException,
java.rmi.RemoteException
```

getRoleAssignmentRequestStatus

Returns a list of role assignment request status instances given a correlation ID.

Syntax: Here is the method signature:

```
RoleAssignmentRequestStatusArray
getRoleAssignmentRequestStatus(java.lang.String correlationId) throws
NrfServiceException, java.rmi.RemoteException
```

getRoleAssignmentRequestStatusByIdentityType

Returns a list of role assignment request status instances given an identity and an identity type.

Syntax: Here is the method signature:

```
RoleAssignmentRequestStatusArray
getRoleAssignmentRequestStatusByIdentityType(java.lang.String identityDn,
IdentityType identityType) throws NrfServiceException,
java.rmi.RemoteException
```

getRoleAssignmentTypeInfo

Retrieves details about a RoleAssignmentType.

Syntax: Here is the method signature:

```
RoleAssignmentTypeInfo getRoleAssignmentTypeInfo(RoleAssignmentType type)
throws NrfServiceException, java.rmi.RemoteException
```

getRoleCategories

Gets role categories.

Syntax: Here is the method signature:

```
CategoryArray getRoleCategories() throws NrfServiceException,
java.rmi.RemoteException
```

getRoleConflicts

Returns a list of Sod instances found for all given roles. This method always returns a list.

Syntax: Here is the method signature:

```
SodArray getRoleConflicts(DNStringArray roles) throws NrfServiceException,
java.rmi.RemoteException
```

getRoleLevels

Gets role levels.

Syntax: Here is the method signature:

```
RoleLevelArray getRoleLevels() throws NrfServiceException,
java.rmi.RemoteException
```

getRolesInfo

Returns a list of RoleInfo instances given a list of role DNs.

Syntax: Here is the method signature:

```
RoleInfoArray getRolesInfo(DNStringArray roleDns) throws
NrfServiceException, java.rmi.RemoteException
```

getRolesInfoByCategory

Returns a list of RoleInfo instances given a list of role category keys.

Syntax: Here is the method signature:

```
RoleInfoArray getRolesInfoByCategory(CategoryKeyArray roleCategoryKeys)
throws NrfServiceException, java.rmi.RemoteException
```

getRolesInfoByLevel

Returns a list of RoleInfo instances given a list of role levels.

Syntax: Here is the method signature:

```
RoleInfoArray getRolesInfoByLevel(LongArray roleLevels) throws
NrfServiceException, java.rmi.RemoteException
```

getTargetSourceConflicts

Returns a list of Sod instances for all SOD conflicts defined between the target role DN and the source role DN.

Syntax: Here is the method signature:

```
SodArray getTargetSourceConflicts(java.lang.String targetName,
java.lang.String sourceName) throws NrfServiceException,
java.rmi.RemoteException
```

getUser

Gets user info including all role assignments for a given user DN stored in a UserIdentity object.

Syntax: Here is the method signature:

```
User getUser(java.lang.String userDn) throws NrfServiceException,
java.rmi.RemoteException
```

getVersion

Returns the version of this Web Service.

Syntax: Here is the method signature:

```
VersionVO getVersion() throws java.rmi.RemoteException
```

isUserInRole

Returns boolean flag; true if role has been assigned to a User identity

Syntax: Here is the method signature:

```
boolean isUserInRole(java.lang.String userDn, java.lang.String roleDn)
```

requestRoleAssignment

Returns a list of request DNs created by the role assignment

Syntax: Here is the method signature:

```
DNStringArray requestRolesAssignment(RoleAssignmentRequest
roleAssignmentRequest) throws NrfServiceException,
java.rmi.RemoteException
```

Sod

Value object to hold SOD information.

Sod constructors

The Sod class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
Sod()
```

getApprovalType

Gets the SOD approval type.

Syntax: Here is the method signature:

```
public SodApprovalType getApprovalType()
```

getApprovers

Gets SOD approvers.

Syntax: Here is the method signature:

```
public ApproverArray getApprovers()
```

getDescription

Gets the SOD description.

Syntax: Here is the method signature:

```
public java.lang.String getDescription()
```

getEntityKey

Gets the SOD entity key.

Syntax: Here is the method signature:

```
public java.lang.String getEntityKey()
```

getName

Gets the SOD name.

Syntax: Here is the method signature:

```
public java.lang.String getName()
```

getQuorum

Gets the SOD quorum amount.

Syntax: Here is the method signature:

```
public java.lang.String getQuorum()
```

getRequestDef

Gets the request definition for approval processing.

Syntax: Here is the method signature:

```
public java.lang.String getRequestDef()
```

getRoles

Gets the SOD roles.

Syntax: Here is the method signature:

```
public DNStringArray getRoles()
```

setApprovalType

Sets the SOD approval type.

Syntax: Here is the method signature:

```
public void setApprovalType(SodApprovalType approvalType)
```

setApprovers

Sets the SOD approvers.

Syntax: Here is the method signature:

```
public void setApprovers(ApproverArray approvers)
```

setDescription

Sets the SOD description.

Syntax: Here is the method signature:

```
public void setDescription(java.lang.String description)
```

setEntityKey

Sets the SOD entity key.

Syntax: Here is the method signature:

```
public void setEntityKey(java.lang.String entityKey)
```

setName

Sets the SOD name.

Syntax: Here is the method signature:

```
public void setName(java.lang.String name)
```

setQuorum

Sets the SOD quorum amount.

Syntax: Here is the method signature:

```
public void setQuorum(java.lang.String quorum)
```

setRequestDef

Sets the request definition for approval processing.

Syntax: Here is the method signature:

```
public void setRequestDef(java.lang.String requestDef)
```

setRoles

Sets the SOD roles.

Syntax: Here is the method signature:

```
public void setRoles(DNStringArray roles)
```

SodArray

This section provides reference information on the SodArray class.

SodArray constructors

The SodArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
SodArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of Attribute objects as a parameter:

```
SodArray(Sod[] SodVal)
```

getSod

Returns an array of Sod objects.

Syntax: Here is the method signature:

```
Sod[] getSod()
```

setSod

Sets the array of Sod objects associated with the SodArray class.

Syntax: Here is the method signature:

```
void setSod (Sod[] SodVal)
```

SodApprovalType

An JAX-RPC friendly representation of com.novell.idm.nrf.api.SodApprovalType.

Table 29-8 *Field Summary*

Type	Name
static SodApprovalType	ALLOW_WITH_WORKFLOW
static SodApprovalType	ALWAYS_ALLOW

SodApprovalType constructors

The SodApprovalType class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
SodApprovalType()
```

Syntax 2: Here is the syntax for a constructor that takes a String as a parameter:

```
SodApprovalType(java.lang.String value)
```

equals

Implementation of equals().

Syntax: Here is the method signature:

```
public boolean equals(java.lang.Object obj)
```

fromRPC

Reconstructs an API representation object from an RPC representation.

Syntax: Here is the method signature:

```
public com.novell.idm.nrf.api.SodApprovalType fromRPC() throws
com.novell.idm.nrf.exception.NrfException
```

fromValue

This method is for WSSDK serialization.

Syntax: Here is the method signature:

```
public static SodApprovalType fromValue(java.lang.String value)
```

getValue

Gets the type.

Syntax: Here is the method signature:

```
public java.lang.String getValue()
```

hashCode

This is an implementation of hashCode(). This implementation overrides the hashCode() method in java.lang.Object.

Syntax: Here is the method signature:

```
public int hashCode()
```

setValue

Sets the type.

Syntax: Here is the method signature:

```
public void setValue(java.lang.String type)
```

toRPC

Reconstructs an API representation object from an RPC representation.

Syntax: Here is the method signature:

```
public com.novell.idm.nrf.api.SodApprovalType fromRPC() throws
com.novell.idm.nrf.exception.NrfException
```

toString

Implementation of toString() that returns a string representation of the class.

Syntax: Here is the method signature:

```
public java.lang.String toString()
```

SodJustification

Class to represent an SOD DN to override with a justification. Used for assignment of roles to be able to pass in a justification for overrides of SODs.

SodJustification constructors

The SodJustification class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
SodJustification()
```

Syntax 2: Here is the syntax for a constructor that takes two String values as parameters:

```
SodJustification(java.lang.String sodDN, java.lang.String justification)
```

getJustification

Gets the SOD justification for override.

Syntax: Here is the method signature:

```
public java.lang.String getJustification()
```

getSodDN

Gets the SOD DN for override.

Syntax: Here is the method signature:

```
public java.lang.String getSodDN()
```

setJustification

Sets the justification for override.

Syntax: Here is the method signature:

```
public void setJustification(java.lang.String justification)
```

setSodDN

Sets the SOD DN for override.

Syntax: Here is the method signature:

```
public void setSodDN(java.lang.String sodDN)
```

SodJustificationArray

This section provides reference information on the SodJustificationArray class.

SodJustificationArray constructors

The SodJustificationArray class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
SodJustificationArray()
```

Syntax 2: Here is the syntax for a constructor that takes an array of Attribute objects as a parameter:

```
SodJustificationArray(SodJustification[] SodJustificationVal)
```

getSodjustification

Returns an array of SodJustification objects.

Syntax: Here is the method signature:

```
SodJustification[] getSodjustification()
```

setSodjustification

Sets the array of SodJustification objects associated with the SodJustificationArray class.

Syntax: Here is the method signature:

```
void setSodjustification (SodJustification[] SodJustificationVal)
```

User

Value class to hold user identity information.

User constructors

The User class supports a single constructor.

Syntax: Here is the syntax for the constructor:

```
User()
```

getAssociatedRoles

Gets the associated roles for this identity.

Syntax: Here is the method signature:

```
public DNStringArray getAssociatedRoles()
```

getCn

Gets the cn.

Syntax: Here is the method signature:

```
public java.lang.String getCn()
```

getContainerRoles

Gets the container roles.

Syntax: Here is the method signature:

```
public DNStringArray getContainerRoles()
```

getEmail

Gets the email address.

Syntax: Here is the method signature:

```
public java.lang.String getEmail()
```

getEntityKey

Gets the identity entity key.

Syntax: Here is the method signature:

```
public java.lang.String getEntityKey()
```

getExplicitAssignments

Gets the explicit role assignments.

Syntax: Here is the method signature:

```
public RoleAssignmentArray getExplicitAssignments()
```

getFirstName

Gets the first name.

Syntax: Here is the method signature:

```
public java.lang.String getFirstName()
```

getGroupRoles

Gets the group roles.

Syntax: Here is the method signature:

```
public DNStringArray getGroupRoles()
```

getIdentityType

Gets identity type.

Syntax: Here is the method signature:

```
public IdentityType getIdentityType()
```

getImplicitAssignments

Gets the implicit role assignments.

Syntax: Here is the method signature:

```
public RoleAssignmentArray getImplicitAssignments()
```

getInheritedAssignments

Gets the inherited role assignments.

Syntax: Here is the method signature:

```
public RoleAssignmentArray getInheritedAssignments()
```

getInheritedRoles

Gets the inherited roles.

Syntax: Here is the method signature:

```
public DNStringArray getInheritedRoles()
```

getLastName

Gets the last name.

Syntax: Here is the method signature:

```
public java.lang.String getLastName()
```

getRoleAssignments

Gets the role assignments for this identity.

Syntax: Here is the method signature:

```
public RoleAssignmentArray getRoleAssignments()
```

setAssociatedRoles

Sets the associated roles for this identity.

Syntax: Here is the method signature:

```
public void setAssociatedRoles(DNStringArray associatedRoles)
```

setCn

Sets the CN.

Syntax: Here is the method signature:

```
public void setCn(java.lang.String cn)
```

setContainerRoles

Sets the container roles.

Syntax: Here is the method signature:

```
public void setContainerRoles(DNStringArray containerRoles)
```

setEmail

Sets the email address.

Syntax: Here is the method signature:

```
public void setEmail(java.lang.String email)
```

setEntityKey

Sets the identity entity key.

Syntax: Here is the method signature:

```
public void setEntityKey(java.lang.String entityKey)
```

setExplicitAssignments

Sets the explicit role assignments.

Syntax: Here is the method signature:

```
public void setExplicitAssignments(RoleAssignmentArray
explicitAssignments)
```

setFirstName

Sets the first name.

Syntax: Here is the method signature:

```
public void setFirstName(java.lang.String firstName)
```

setGroupRoles

Sets the group roles.

Syntax: Here is the method signature:

```
public void setGroupRoles(DNStringArray groupRoles)
```

setIdentityType

Sets the identity type.

Syntax: Here is the method signature:

```
public void setIdentityType(IdentityType identityType)
```

setImplicitAssignments

Sets the implicit role assignments.

Syntax: Here is the method signature:

```
public void setImplicitAssignments(RoleAssignmentArray
implicitAssignments)
```

setInheritedAssignments

Sets the inherited role assignments.

Syntax: Here is the method signature:

```
public void setInheritedAssignments(RoleAssignmentArray
inheritedAssignments)
```

setInheritedRoles

Sets the inherited roles.

Syntax: Here is the method signature:

```
public void setInheritedRoles(DNStringArray inheritedRoles)
```

setLastName

Sets the last name.

Syntax: Here is the method signature:

```
public void setLastName(java.lang.String lastName)
```

setRoleAssignments

Sets the role assignments for this identity.

Syntax: Here is the method signature:

```
public void setRoleAssignments(RoleAssignmentArray roleAssignments)
```

VersionVO

A value object for Version.

VersionVO constructors

The VersionVO class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
VersionVO()
```

Syntax 2: Here is the syntax for a constructor that takes a String as a parameter:

```
VersionVO(java.lang.String version)
```

getValue

Gets the version.

Syntax: Here is the method signature:

```
public java.lang.String getValue()
```

setValue

Sets the version.

Syntax: Here is the method signature:

```
public void setValue(java.lang.String version)
```

Role Web Service Examples

This section provides examples that demonstrate how you might use the Role service.

Retrieving Roles for a Group

This example shows how to retrieve the role assignments for a given group:

```
public void getGroupTestCase()
    throws Exception
  {
    System.out.println("\n***************Calling
getGroupTestCase()*******************************");
    String groupDN = "cn=HR,ou=groups,ou=medical-idmsample,o=netiq";
    try
    {
      IRemoteRole stub = getRolesStub(username, password, acceptlanguage);
      Group group = stub.getGroup(groupDN);
      //Assert.assertNotNull("Group not found", group);
      if (group != null)
      {
        System.out.println("Group Found:");
        System.out.println("   entityKey          : " +
group.getEntityKey());
        System.out.println("   identityType       : " +
group.getIdentityType().getValue());
        System.out.println("   description        : " +
group.getDescription());

        DNString[] roles = group.getAssociatedRoles().getDnstring();
        if (roles != null)
        {
          System.out.println("no of associated roles: " + roles.length);
          for (int rIndex = 0; rIndex < roles.length; rIndex++)
          {
            System.out.println("      role: " + rIndex);
          }
```

```
        }
        else
        {
          System.out.println("no of associated roles:0");
        }

        RoleAssignment[] assignments =
group.getRoleAssignments().getRoleassignment();
        PrintRoleUtils.getAssignments(assignments);
      }
      else
        System.out.println("Group not found");
    }
    catch (NrfServiceException nrf)
    {
      throw new Exception(nrf.getMessage());
    }
    catch (RemoteException re)
    {
      throw new Exception(re.getMessage());
    }
  }

...
 /**
     * Returns the Roles remote stub
     * @param username - user name
     * @param password - password
     * @param acceptLanguage - HTTP header Accept-Language
     * @return the Roles remote stub
     * @throws Exception - catch all exceptions
     */
    public static IRemoteRole getRolesStub(String username,
                                           String password,
                                           String acceptLanguage)
        throws Exception
    {
        Stub stub = null;
        String stubCacheKey = username + ":" + password;
        if (g_rolesStubCache.containsKey(stubCacheKey)) {
            g_log.debug("Using Cached Roles stub for [" + username + "]");
            stub = (Stub) g_rolesStubCache.get(stubCacheKey);
        } else {
            g_log.debug("Using New Roles stub");
            RoleService service = new RoleServiceImpl();
            stub = (Stub) service.getIRemoteRolePort();

            if (username != null && password != null) {
                stub._setProperty(Stub.USERNAME_PROPERTY, username);
                stub._setProperty(Stub.PASSWORD_PROPERTY, password);
            }
```

```
            stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY,
ServletParameters.getInstance().getUserAppUrl() + ROLES_SERVICE);
            stub._setProperty(Stub.SESSION_MAINTAIN_PROPERTY,
Boolean.TRUE);

            g_rolesStubCache.put(stubCacheKey, stub);
        }

        Properties props = new Properties();
        props.setProperty("Accept-Language", acceptLanguage);
        stub._setProperty(Stub.HTTP_HEADERS, props);

        return (IRemoteRole) stub;
    }
```

Retrieving Role Assignment Request Status

Returns a list of role assignment request status instances given a correlation ID.

```
  public void getRoleAssignmentRequestStatusTestCase()
    throws Exception
  {
    System.out.println("\n***************Calling


getRoleAssignmentRequestStatusTestCase()*******************************")
;
    String correlationId = "9a5feec728864b55ac443724a915e831";
    try
    {
      IRemoteRole stub = getRoleStub(url, username, password);
      RoleAssignmentRequestStatusArray reqArray =
stub.getRoleAssignmentRequestStatus(correlationId);
      RoleAssignmentRequestStatus[] reqStatus =
reqArray.getRoleassignmentrequeststatus();
      //Assert.assertNotNull("RoleAssignmentRequestStatus object is null
for


getRoleAssignmentRequestStatus", reqStatus);
      if (reqStatus != null)
        System.out.println(PrintRoleUtils.getRequestStatus(reqStatus));
      else
       System.out.println("RoleAssignmentRequestStatus object is null for
```

```
getRoleAssignmentRequestStatus");

      //result += Util.getRequestStatus(reqStatus);
    }
    catch (NrfServiceException nrf)
    {
      throw new Exception(nrf.getMessage());
    }
    catch (RemoteException re)
    {
      throw new Exception(re.getMessage());
    }

  }
```

Retrieving Type Information for a Role Assignment

This example shows how to retrieve the type for a role assignment:

```
  public void getRoleAssignmentTypeInfoTestCase()
    throws Exception
  {
    System.out.println("\n***************Calling


getRoleAssignmentTypeInfoTestCase()*******************************");
    try
    {
      IRemoteRole stub = getRoleStub(url, username, password);

      RoleAssignmentTypeInfo info =

stub.getRoleAssignmentTypeInfo(RoleAssignmentType.fromValue("ROLE_TO_ROLE"
));
      //Assert.assertNotNull("Role Assignment Type Info Not Found for
getRoleAssignmentTypeInfo", info);
      if (info != null)
      {
        System.out.println("Role Assignment Type Info:");
        System.out.println("           identity type: " +
info.getIdentityType().getValue());
        System.out.println("         subtree included: " +
info.getSubtreeIncluded());
        System.out.println("        suports approvals: " +
info.getSupportsApproval());
        System.out.println("  supports effective date: " +
info.getSupportsEffectiveDate());
        System.out.println("      supports expiration: " +
info.getSupportsExpiration());
        System.out.println("    supports SOD Approval: " +
info.getSupportsSODApproval());
      }
```

```
      else
        System.out.println("Role Assignment Type Info Not Found for
getRoleAssignmentTypeInfo");
    }
    catch (NrfServiceException nrf)
    {
      throw new Exception(nrf.getMessage());
    }
    catch (RemoteException re)
    {
      throw new Exception(re.getMessage());
    }

  }
```

Retrieving Role Categories

This example shows how to retrieve the defined role categories:

```
  public void getRoleCategoriesTestCase()
    throws Exception
  {
    System.out.println("\n***************Calling
getRoleCategoriesTestCase()*******************************");
    try
    {
      IRemoteRole stub = getRoleStub(url, username, password);
      CategoryArray entriesArray = stub.getRoleCategories();
      Category[] entries = entriesArray.getCategory();
      Assert.assertNotNull("No categories found.", entries);
      if (entries != null)
      {
        System.out.println("no of categories:" + entries.length);

        for (int i = 0; i < entries.length; i++)
        {
          System.out.println("   category key  : " +
entries[i].getCategoryKey());
          System.out.println("   category label: " +
entries[i].getCategoryLabel());
        }
      }
      else
        System.out.println("No categories found.");
    }
    catch (NrfServiceException nrf)
    {
      throw new Exception(nrf.getMessage());
    }
    catch (RemoteException re)
    {
      throw new Exception(re.getMessage());
    }

  }
```

Retrieving Role Levels

This example shows how to retrieve the defined role levels:

```
  public void getRoleLevelsTestCase()
    throws Exception
  {
    System.out.println("\n***************Calling
getRoleLevelsTestCase()*******************************");
    try
    {
      IRemoteRole stub = getRoleStub(url, username, password);
      RoleLevelArray roleLevelArray = stub.getRoleLevels();
      RoleLevel[] entries = roleLevelArray.getRolelevel();
      //Assert.assertNotNull("No role levels found.", entries);
      if (entries != null)
      {
        System.out.println("no of levels:" + entries.length);

        for (int index = 0; index < entries.length; index++)
        {
          System.out.println("   Level      : " +
entries[index].getLevel());
         System.out.println("   Name       : " + entries[index].getName());
          System.out.println("   Description: " +
entries[index].getDescription());
          System.out.println("   Container  : " +
entries[index].getContainer());
        }
      }
      else
        System.out.println("No role levels found.");
    }
    catch (NrfServiceException nrf)
    {
      throw new Exception(nrf.getMessage());
    }
    catch (RemoteException re)
    {
      throw new Exception(re.getMessage());
    }

  }
```

Verifying Whether a User Is In a Role

This example shows how to determine whether a user has been assigned to a role:

```
  public void isUserInRoleTestCase()
    throws Exception
  {
    System.out.println("\n***************Calling
isUserInRoleTestCase()*******************************");
    String[] DNs = {
                "cn=ablake,ou=users,ou=medical-idmsample,o=netiq",

"cn=Doctor,cn=Level20,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=HajenDrive
r,cn=TestDrivers,o=netiq"
                };
    try
    {
      IRemoteRole stub = getRoleStub(url, username, password);
      boolean inRole = stub.isUserInRole(DNs[0], DNs[1]);

      String sInRole = "User Not In Role";
      if (inRole)
        sInRole = new String("User In Role");

      System.out.println(sInRole);
    }
    catch (NrfServiceException nrf)
    {
      throw new Exception(nrf.getMessage());
    }
    catch (RemoteException re)
    {
      throw new Exception(re.getMessage());
    }

  }
```

30 Resource Web Service

This section describes the Resource Web Service, which allows SOAP clients to invoke a subset of actions that apply to resources.

About the Resource Web Service

The Resource Web Service exposes a small set of actions for the resource model. The service allows remote clients to request that a resource be granted or revoked, and also to check on the status of resource requests. By exposing these actions, the service makes it possible for a provisioning workflow to invoke resource requests through the Integration activity.

Calls to the Resource Web Service calls require HTTP authentication. By default, access to the resource service methods is restricted to Resource Administrators.

Accessing the Test Page

You can access the Resource Web Service endpoint using a URL similar to the following:

`http://server:port/warcontext/resource/service?test`

For example, if your server is named “myserver”, your User Application is listening on port 8080, and your User Application war file is named “IDMPROV”, the URL would be:

```
http://myserver:8080/IDMPROV/resource/service?test
```

WARNING: The test page is disabled by default. Since some of the methods allow data to be updated, the test page presents a potential security vulnerability and should not be allowed in a production environment.

Servlet declaration for the Resource Service

A SOAP service using WSSDK is deployed by adding the following declarations in the deployment descriptor (i.e. WEB-INF/web.xml):

```
<servlet>
  <servlet-name>Resource</servlet-name>
  <servlet-
class>com.novell.idm.nrf.soap.ws.role.impl.ResourceServiceSkeletonImpl</
servlet-class>

<servlet-mapping>
  <servlet-name>Resource</servlet-name>
  <url-pattern>/resource/service</url-pattern>
</servlet-mapping>
</servlet>
```

This follows the normal servlet declaration pattern. It indicates that the servlet com.novell.idm.nrf.soap.ws.resource.impl.ResourceServiceSkeletonImpl is deployed at /resource/service.

When a user reaches this servlet using a HTTP GET by entering `http://server-name/context/resource/service` (for example, `http://localhost:8080/IDMProv/resource/service`) in their browser, the WSSDK provides a page that exposes some information about the deployed service.

Enabling the Test Page

WARNING: The test page is disabled by default. Since some of the methods allow data to be updated, the test page presents a potential security vulnerability and should not be allowed in a production environment.

To enable the test page, you need to update the WEB-INF/web.xml file in the IDMProv.war file. Before you make your changes, the web.xml should look like this:

```
<servlet>
  <servlet-name>Resource</servlet-name>
  <servlet-
class>com.novell.idm.nrf.soap.ws.resource.impl.ResourceServiceSkeletonImpl
</servlet-class>
  <init-param>
    <param-name>com.novell.soa.ws.test.disable</param-name>
    <param-value>true</param-value>
  </init-param>
</servlet>
```

Change the servlet declaration, as follows:

```
<servlet>
  <servlet-name>Resource</servlet-name>
  <servlet-
class>com.novell.idm.nrf.soap.ws.resource.impl.ResourceServiceSkeletonImpl
</servlet-class>
  <init-param>
    <param-name>com.novell.soa.ws.test.disable</param-name>
    <param-value>false</param-value>
  </init-param>
</servlet>
```

Accessing the WSDL

You can access the WSDL for the Resource Web Service using a URL similar to the following:

`http://server:port/warcontext/resource/service?wsdl`

For example, if your server is named “myserver”, your User Application is listening on port 8080, and your User Application war file is named “IDMPROV”, the URL would be:

`http://myserver:8080/IDMPROV/resource/service?wsdl`

Generating the Stub Classes

Before using the Web Service, you need to use the WSSDK tool or another SOAP tool kit to generate the stub classes. To allow your code to find the stub classes, you also need to add the JAR that contains the stub classes to your classpath.

If you want to use the NetIQ WSSDK tool, you can generate the client stubs by extracting the WSDL and running the wsdl2java utility. For example, you could run this command to generate the stubs in a package called com.novell.soa.af.resource.soap.impl:

```
"C:\Program Files\Java\jdk1.6.0_31\bin\java" -cp "../lib/wssdk.jar;../lib/
jaxrpc-api.jar";"../lib/mail.jar";"../lib/activation.jar";"c:\Program
Files\Java\jdk1.6.0_31\lib\tools.jar";
com.novell.soa.ws.impl.tools.wsdl2java.Main -verbose -ds gensrc -d C:\ -
noskel -notie -genclient -keep -package
com.novell.soa.af.resource.soap.impl -javadoc resource.wsdl
```

You can change the wsdl2java parameters to suit your requirements.

Removing Administrator Credential Restrictions

The Resource Web Service supports two levels of security, one that restricts access to Resource Administrators, and another that restricts access to the authenticated user. The default setting restricts access to all operations to the Resource Administrator.

You can modify the settings for security configuration in the `ism-configuration.properties` file, located by default in the `/netiq/idm/apps/tomcat/conf` directory. Each property can be set to true or false. A value of true locks down the operation, whereas a value of false opens up the operation.

You can open up the Resource Web Service to authenticated users by setting the ResourceService/Resource/soap property to false. To open up a particular operation to authenticated users, you need to set the property for that operation (ResourceService/Resource/soap/*operation*) to false as well. If you set all of the properties to false, you can open up all operations to authenticated users. The *operation* names are the same as the names of the methods supported by the service.

The following methods can be invoked by users without Resource Administrator credentials if the property ResourceService/Resource/soap property is set to false:

- requestResourceGrant
- requestResourceRevoke
- getResourceRequestStatusByCorrelationId
- getResourceRequestStatusForCurrentUser
- getResourceAssignmentsForCurrentUser

If you wish to change the restriction for a particular operation, you can modify the property ResourceService/Resource/soap/operation for the method, setting its value to true to restrict access to administrators for the operation and false to remove the restriction. If the ResourceService/Resource/soap property is true, all methods are restricted to Resource Administrator credentials.

Example To ensure that the security configuration opens up all operations within the Resource Web Service, except for the getResourceRequestStatusByIdentity operation, which would only be accessible to the Resource Administrator, the `ism-configuration.properties` must have the following settings:

```
  ResourceService/Resource/soap = false
   ResourceService/Resource/soap/requestResourceGrant = false
   ResourceService/Resource/soap/requestResourceRevoke = false
   ResourceService/Resource/soap/getResourceRequestStatusByCorrelationId =
false
   ResourceService/Resource/soap/getResourceRequestStatusForCurrentUser =
false
   ResourceService/Resource/soap/getResourceRequestStatusByIdentity = true
```

Resource Web Service Interface

This section provides details about the methods available with the Resource Web service. This programming interface presumes you're using Java code generated by the WSSDK toolkit. The interface will be different if you're using another Web Service toolkit.

IRemoteResource

This section provides reference information for each method associated with the IRemoteResource interface.

checkCodeMapValueStatus

Checks to see if a particular value exists in the code map table for a specified entitlement and logical system. The method returns the status for the code map value as a CodeMapValueStatus object.

This method is one of three SOAP endpoints to help you keep the code map tables for the Roles Based Provisioning Module synchronized with the code map tables for the Role Mapping Administrator. The user interface for the Role Mapping Administrator can trigger a code map refresh if a mismatch is discovered while a user is creating mappings. In addition, the Roles Based Provisioning Module allows you to use the three SOAP endpoints to refresh selected entitlements within its code map tables.

In addition to checkCodeMapValueStatus, the Roles Based Provisioning Module includes the following endpoints to help with code map synchronization:

- getRefreshStatus
- refreshCodeMap

The **Entitlement Query Settings** section of the **Configure Roles and Resources Settings** page in the User Application allows you to specify how often the Roles Based Provisioning Module code map tables are refreshed and also start a manual refresh. However, this page does not allow to refresh selected entitlements. To control which entitlements are refreshed, you need to use the SOAP endpoints.

For additional information on the getRefreshStatus endpoint, see "getRefreshStatus" on page 493. For additional information on the refreshCodeMap endpoint, see "refreshCodeMap" on page 499.

For code samples that use the new methods for code map synchronization, see "Code Map Synchronization Code Samples" on page 520.

Syntax: Here is the method signature:

```
public CodeMapValueStatus checkCodeMapValueStatus(String entitilementDN,
String connectionName, String codeMapValue)
       throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *entitlementDN* entitlement DN as a string.

  For example:

  ```
  cn=groups,cn=groupentitlementloopback,cn=driverset1,o=system
  ```

- *connectionName* connection (logical system) name. This is an optional parameter. Only fanout drivers need to specify the connection name.
- *codeMapValue* code map value to verify.

  For example:

  ```
  \TEST1\data\groups\netiq\cambridge\rbpm\4AlphaGroup
  ```

SOAP Request: Here is the SOAP request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
" xmlns:ser="http://www.netiq.com/resource/service">
<soapenv:Header/>
<soapenv:Body>
<ser:checkCodeMapValueStatusRequest>
<!--Optional:-->
<ser:entitilementDN>cn=groups,cn=groupentitlementloopback,cn=driverset1,o=
system</ser:entitilementDN>
<!--Optional:-->
<ser:connectionName/>
<!--Optional:-->
<ser:codeMapValue>\WILLIAMS1\data\groups\netiq\cambridge\rbpm\4AlphaGroup<
/ser:codeMapValue>
</ser:checkCodeMapValueStatusRequest>
</soapenv:Body>
</soapenv:Envelope>
```

SOAP Response: Here is the SOAP response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body>
<ns1:checkCodeMapValueStatusResponse xmlns="http://www.netiq.com/resource/
service" xmlns:ns1="http://www.netiq.com/resource/service">
<result>
<refreshStatus>
<connectionName xsi:nil="1"/>
<entitlementDN>cn=groups,cn=groupentitlementloopback,cn=driverset1,o=syste
m</entitlementDN>
<guid>\2d\13\d1\a4\7b\99\d6\4c\03\9a\2d\13\d1\a4\7b\99</guid>
<lastRefresh>1329431650891</lastRefresh>
<status>SUCCESS</status>
</refreshStatus>
<upToDate>true</upToDate>
<value>\WILLIAMS1\data\groups\netiq\cambridge\rbpm\4AlphaGroup</value>
</result>
</ns1:checkCodeMapValueStatusResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

createResource

Creates a new resource according to the specified parameters, and returns a DN of the created resource.

A correlation ID is generated automatically for this method that uses this format:

`UserApp#RemoteResourceRequest#xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`

The correlation ID is used for auditing.

Syntax: Here is the method signature:

```
public String createResource(Resource resource)
            throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *resource* specifies the resource object to create.

createResourceAid

Creates a new resource, with a correlation ID that you provide. The correlation ID is used for auditing to link a set of related resources. This method creates the resource according to the specified parameters, and returns a DN of the created resource.

Syntax: Here is the method signature:

```
public String createResourceAid(Resource resource, String correlationId)
            throws NrfServiceException, RemoteException;
```

findResourceByExampleWithOperator

Finds all Resource objects based on the search criteria specified in the given Resource object.

Syntax: Here is the method signature:

```
public Resource[] findResourceByExampleWithOperator(Resource
searchCriteria, boolean useAndForMultiValueSearch)
            throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *searchCriteria* specifies Query by Example (QBE) search criteria within a Resource object.
- *useAndForMultiValueSearch* determines whether AND or OR will be used for multi-value search expressions. If you specify a value of true, AND will be used for multi-value searches; if you specify a value of false, OR will be used.

getEntitlementCodeMap

Returns an array of ProvisioningCodeMap objects, which include code map information from the code map and code map label tables.

Syntax: Here is the method signature:

```
ProvisioningCodeMap[] getEntitlementCodeMap(java.lang.String codeMapKey,
int type)
        throws com.novell.idm.nrf.soap.ws.resource.NrfServiceException,
java.rmi.RemoteException;
```

The parameters are described below:

- *codeMapKey* specifies the code map key to retrieve values from. The codeMapKey is a GUID that acts as a unique identifier for the code map. For example:

  ```
  \2d\13\d1\a4\7b\99\d6\4c\03\9a\2d\13\d1\a4\7b\99
  ```

- *type* specifies the code map type. A value of 0 filters the list to include entitlement code maps only.

getRefreshStatus

Gets the refresh status of a code map based on a specified entitlement DN. This method returns the status as an array of CodeMapRefreshStatus objects. The structure returned contains the DN, GUID, connection name status, and last refresh time.

This method is one of three SOAP endpoints to help you keep the code map tables for the Roles Based Provisioning Module synchronized with the code map tables for the Role Mapping Administrator. The user interface for the Role Mapping Administrator can trigger a code map refresh if a mismatch is discovered while a user is creating mappings. In addition, the Roles Based Provisioning Module allows you to use the three SOAP endpoints to refresh selected entitlements within its code map tables.

In addition to getRefreshStatus, the Roles Based Provisioning Module includes the following endpoints to help with code map synchronization:

- checkCodeMapValueStatus
- refreshCodeMap

The **Entitlement Query Settings** section of the **Configure Roles and Resources Settings** page in the User Application allows you to specify how often the Roles Based Provisioning Module code map tables are refreshed and also start a manual refresh. However, this page does not allow to refresh selected entitlements. To control which entitlements are refreshed, you need to use the SOAP endpoints.

For additional information on the checkCodeMapValueStatus endpoint, see "checkCodeMapValueStatus" on page 490. For additional information on the refreshCodeMap endpoint, see "refreshCodeMap" on page 499.

For code samples that use the new methods for code map synchronization, see "Code Map Synchronization Code Samples" on page 520.

Syntax: Here is the method signature:

```
public CodeMapRefreshStatus[] getRefreshStatus(String entitlementDN)
          throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *entitlementDN* entitlement DN as a string

  For example:

  ```
  cn=groups,cn=groupentitlementloopback,cn=driverset1,o=system
  ```

SOAP Request: Here is the SOAP request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
" xmlns:ser="http://www.netiq.com/resource/service">
<soapenv:Header/>
<soapenv:Body>
<ser:getRefreshStatusRequest>
<!--Optional:-->
<ser:entitlementDN>cn=groups,cn=groupentitlementloopback,cn=driverset1,o=s
ystem</ser:entitlementDN>
</ser:getRefreshStatusRequest>
</soapenv:Body>
</soapenv:Envelope>
```

SOAP Response: Here is the SOAP response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body>
<ns1:getRefreshStatusResponse xmlns="http://www.netiq.com/resource/
service" xmlns:ns1="http://www.netiq.com/resource/service">
<result>
<codemaprefreshstatus>
<connectionName xsi:nil="1"/>
<entitlementDN>cn=groups,cn=groupentitlementloopback,cn=driverset1,o=syste
m</entitlementDN>
<guid>\2d\13\d1\a4\7b\99\d6\4c\03\9a\2d\13\d1\a4\7b\99</guid>
<lastRefresh>1329100366090</lastRefresh>
<status>SUCCESS</status>
</codemaprefreshstatus>
</result>
</ns1:getRefreshStatusResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

getResourceAssignmentsForCurrentUser

Returns the resource assignments for the current user.

Syntax: Here is the method signature:

```
ResourceAssignment[] getResourceAssignmentsForCurrentUser()
        throws com.novell.idm.nrf.soap.ws.resource.NrfServiceException,
java.rmi.RemoteException;
```

getResourceAssignmentsForUser

Returns the resource assignments for a particular user.

Syntax: Here is the method signature:

```
ResourceAssignment[] getResourceAssignmentsForUser(java.lang.String
userDn)
        throws com.novell.idm.nrf.soap.ws.resource.NrfServiceException,
java.rmi.RemoteException;
```

The parameters are described below:

- *userDn* DN of the target user

getAssignmentsForResource

Returns the resource assignments for a particular resource.

Syntax: Here is the method signature:

```
ResourceAssignment[] getAssignmentsForResource(java.lang.String
resourceDn)
        throws com.novell.idm.nrf.soap.ws.resource.NrfServiceException,
java.rmi.RemoteException;
```

The parameters are described below:

- *resourceDn* DN of the target resource

getResourceRequestStatusByCorrelationId

Returns all resource request status items for a given correlation ID.

Syntax: Here is the method signature:

```
public ResourceAssignmentRequestStatus[]
           getResourceRequestStatusByCorrelationId
                   (String correlationId, String locale)
               throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *correlationId* specifies a resource assignment request correlation ID.
- *locale* supplies an iso639 language code to format localized string values; if the parameter is null, the language defaults to the servlet request locale.

This method returns all resource request status instances for the specified *correlationId* parameter value. For more information on the ResourceAssignmentRequestStatus class, see "ResourceAssignmentRequestStatus" on page 517.

getResourceRequestsStatusForCurrentUser

Returns all resource request status items for the authenticated user.

Syntax: Here is the method signature:

```
 public ResourceAssignmentRequestStatus[]
                   getResourceRequestStatusForCurrentUser(String locale)
               throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *locale* supplies an iso639 language code to format localized string values; if the parameter is null, the language defaults to the servlet request locale.

This method returns all resource request status instances for the specified *correlationId* parameter value. For more information on the ResourceAssignmentRequestStatus class, see "ResourceAssignmentRequestStatus" on page 517.

getResourceRequestStatusByIdentity

Returns all resource assignment request status items for a particular user identity.

Syntax: Here is the method signature:

```
public ResourceAssignmentRequestStatus[]
           getResourceRequestStatusByIdentity(String identity, String
locale)
        throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *identity* specifies the DN for a user.
- *locale* supplies an iso639 language code to format localized string values; if the parameter is null, the language defaults to the servlet request locale.

This method returns all resource request status instances for the specified *correlationId* parameter value. For more information on the ResourceAssignmentRequestStatus class, see "ResourceAssignmentRequestStatus" on page 517.

getCodeMapValues

Returns a list of code map values for a specified code map.

Syntax: Here is the method signature:

```
public CodeMapValue[] getCodeMapValues(String codeMapKey, String locale)
           throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *codeMapKey* specifies the code map key to retrieve values from. The codeMapKey is a GUID that acts as a unique identifier for the code map. For example:

  ```
  \2d\13\d1\a4\7b\99\d6\4c\03\9a\2d\13\d1\a4\7b\99
  ```

- *locale* supplies an iso639 language code to format localized string values; if the parameter is null, the language defaults to the servlet request locale.

getResource

Returns a resource object.

Syntax: Here is the method signature:

```
public Resource getResource(String dn, String locale)
           throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *dn* specifies the DN of the resource you want to retrieve.
- *locale* supplies an iso639 language code to format localized string values; if the parameter is null, the language defaults to the servlet request locale.

getResourceLocalizedStrings

Gets the localized strings for a resource, such as the names and descriptions. The type parameter lets you specify whether the names or descriptions should be retrieved.

Syntax: Here is the method syntax:

```
public LocalizedValue[] getResourceLocalizedStrings(String resourceDn, int
type)throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *resourceDn* specifies the DN of the resource for which you want to get the localized strings.
- *type* specifies the type of localized strings you want to retrieve. A type value of 1 retrieves a list of names for the resource, whereas a type value of 2 retrieves a list of descriptions.

getResourcessInfoByCategory

Returns a list of ResourceInfo instances given a list of category keys.

Syntax: Here is the method signature:

```
public ResourceInfo[] getResourcessInfoByCategory(CategoryKey[]
resourceCategoryKeys)
            throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *resourceCategoryKeys* specifies the list of resource category keys to retrieve resource information objects for.

getResourcessInfo

Returns a list of ResourceInfo instances given a list of resource DNs.

Syntax: Here is the method signature:

```
public ResourceInfo[] getResourcessInfo(DNString[] resDns)
            throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *resDns* provides a list of resource DNs for which you want to retrieve resource information objects.

modifyResource

Modifies a resource definition. This method does not perform a localized string modification update. To update the localized names or descriptions for a resource, you need to use the setResourceLocalizedStrings method.

A correlation ID is generated automatically for this method that uses this format:

```
UserApp#RemoteResourceRequest#xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Syntax: Here is the method signature:

```
public Resource modifyResource(Resource resource)
            throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *resource* specifies the resource object to modify.

modifyResourceAid

Modifies a resource definition, with a correlation ID that you provide. The correlation ID is used for auditing to link a set of related resources. This method does not perform a localized string modification update. To update the localized names or descriptions for a resource, you need to use the setResourceLocalizedStrings method.

Syntax: Here is the method signature:

```
public Resource modifyResourceAid(Resource resource, String correlationId)
           throws NrfServiceException, RemoteException;
```

refreshCodeMap

Refreshes the code map based on a specified entitlement DN. The method returns the status of the refresh operation in the form of an EntitlementRefreshInfo object. This structure includes the detailed status as an array of CodeMapRefreshStatus objects.

This method is one of three SOAP endpoints to help you keep the code map tables for the Roles Based Provisioning Module synchronized with the code map tables for the Role Mapping Administrator. The user interface for the Role Mapping Administrator can trigger a code map refresh if a mismatch is discovered while a user is creating mappings. In addition, the Roles Based Provisioning Module allows you to use the three SOAP endpoints to refresh selected entitlements within its code map tables.

In addition to refreshCodeMap, the Roles Based Provisioning Module includes the following endpoints to help with code map synchronization:

- checkCodeMapValueStatus
- getRefreshStatus

The **Entitlement Query Settings** section of the **Configuration > Roles and Resources** page in the Identity Manager Dashboard allows you to specify how often the Roles Based Provisioning Module code map tables are refreshed and also start a manual refresh.

For additional information on the checkCodeMapValueStatus endpoint, see “checkCodeMapValueStatus” on page 490. For additional information on the getRefreshStatus endpoint, see “getRefreshStatus” on page 493.

For code samples that use the new methods for code map synchronization, see “Code Map Synchronization Code Samples” on page 520.

Syntax: Here is the method signature:

```
public EntitlementRefreshInfo refreshCodeMap(String entitlementDN)
          throws NrfServiceException, RemoteException;
```

The parameter *entitlementDN* entitlement DN is used to refresh the code map:

For example:

```
cn=groups,cn=groupentitlementloopback,cn=driverset1,o=system
```

SOAP Request: Here is the SOAP request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
" xmlns:ser="http://www.netiq.com/resource/service">
<soapenv:Header/>
<soapenv:Body>
<ser:refreshCodeMapRequest>
<!--Optional:-->
<ser:entitlementDN>cn=groups,cn=groupentitlementloopback,cn=driverset1,o=s
ystem</ser:entitlementDN>
</ser:refreshCodeMapRequest>
</soapenv:Body>
</soapenv:Envelope>
```

SOAP Response: Here is the SOAP response:

```
<SOAP-ENV:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
      <ns1:refreshCodeMapResponse xmlns="http://www.netiq.com/resource/
service" xmlns:ns1="http://www.netiq.com/resource/service">
         <result>
            <detailedStatus>
               <codemaprefreshstatus>
                  <connectionName xsi:nil="1"/>

<entitlementDN>cn=groups,cn=groupentitlementloopback,cn=driverset1,o=syste
m</entitlementDN>
                  <guid>\2d\13\d1\a4\7b\99\d6\4c\03\9a\2d\13\d1\a4\7b\99</
guid>
                  <lastRefresh>1329244784180</lastRefresh>
                  <status>SUCCESS</status>
               </codemaprefreshstatus>
            </detailedStatus>

<entitlementDN>cn=groups,cn=groupentitlementloopback,cn=driverset1,o=syste
m</entitlementDN>
            <guid>\2d\13\d1\a4\7b\99\d6\4c\03\9a\2d\13\d1\a4\7b\99</guid>
            <status>true</status>
         </result>
      </ns1:refreshCodeMapResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

removeResource

Deletes a specified resource from the Resource Catalog. Returns the DN for the deleted resource as a confirmation.

A correlation ID is generated automatically for this method that uses this format:

`UserApp#RemoteResourceRequest#xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`

The correlation ID is used for auditing.

Syntax: Here is the method signature:

```
 public DNString removeResource(DNString resourceDn)
            throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *resourceDn* specifies the DN of the resource to delete.

removeResourceAid

Deletes a specified resource from the Resource Catalog, with a correlation ID that you provide. The correlation ID is used for auditing to link a set of related resources. This method returns the DN for the deleted resource as a confirmation.

Syntax: Here is the method signature:

```
 public DNString removeResourceAid(DNString resourceDn, String correlation
Id)
            throws NrfServiceException, RemoteException;
```

requestResourceGrant

Makes a grant resource request and returns a resource request correlation ID.

Syntax: Here is the method signature:

```
public String requestResourceGrant(String resourceTarget, String
requester, String userTarget, String reasonForRequest,
        ResourceRequestParam[] requestParams, String correlationId)
            throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *resourceTarget* specifies the target resource DN.
- *requester* supplies an identifier for the remote client application making the request to grant the resource.

  The *requester* parameter on this SOAP endpoint identifies the originator of the request. This value is set in the resource request object nrfOriginator attribute, following this convention:

  - For a SOAP call: “REMOTE_CLIENT:*<requester param value>*”
  - For a workflow action: “WF:*<wf process id>*”
- *userTarget* specifies the DN for the being granted the resource.
- *reasonForRequest* provides a reason for the request.
- *requestParams* provides the parameter values for the request.
- *correlationId* specifies a resource assignment request correlation ID; if the parameter is null, a correlation ID is generated.

The *requester* parameter is a client-supplied identifier for the agent making the request. For example, an identifier such as *IRemote-MyApplicationName* might be used to identify a request from MyApplicationName. The *requestParams* are the dynamic parameter values required by the resource to make a request. If no values are required, the parameter value can be null or an empty

array. The *correlationId* allows a client to group request for the purpose of checking the staus. If the parameter value is null, the service generates a unique correlation id. The correlation id is returned to the caller.

requestResourceRevoke

Makes a revoke resource request and returns a resource request correlation ID.

The revoke invocation behavior mirrors the behavior for a grant opeation, except that a revoke request for the resource is posted on the server.

Syntax: Here is the method signature:

```
public String requestResourceRevoke(String resourceTarget,
          String requester, String userTarget, String reasonForRequest,
          ResourceRequestParam[] requestParams, String instanceGuid,
String correlationId)
        throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *resourceTarget* specifies the target resource DN.
- *requester* supplies an identifier for the remote client application making the request to revoke the resource.

  The *requester* parameter on this SOAP endpoint identifies the originator of the request. This value is set in the resource request object nrfOriginator attribute, following this convention:

  - For a SOAP call: “REMOTE_CLIENT:*<requester param value>*”
  - For a workflow action: “WF:*<wf process id>*”
  - For the user application user interface: “USER_APP”
- *userTarget* specifies the DN for the user being granted the resource.
- *reasonForRequest* provides a reason for the request.
- *requestParams* provides the parameter values for the request.
- *instanceGuid* provides a GUID identifier for the resource assignment instance. The resource assignment instance GUID supports revoking a single instance of a multi-value resource assignment, if not all instances are to be revoked.

  IMPORTANT: If you do not specify the instanceGuid value, and the user has more than one value of that resource assigned, all instances of the resource assignment will be removed.

  When you create a new resource assignment request, the instanceGuid is included just above the correlationid field:

  ```
  <ser:instanceGuid></ser:instanceGuid>
  ```

  You need to specify the instance of the resource you want to revoke by supplying the value in the *instanceGuid* parameter.

  To find out which resources are assigned to a user, you need to use the getResourceAssignmentsForUser method. This method returns the following data structure, which also includes the instanceGuid:

```
<resourceassignment>
                <instanceGuid>1b335aa9f4a14bd4a2a802eb4ba092da</
instanceGuid>
                <reason>3b-Test</reason>
                <recipientDn>cn=ablake,ou=users,o=data</recipientDn>
                <requestDate>2011-08-18T14:25:21</requestDate>
                <requestParams>
                   <resourcerequestparam>
                      <name>param1</name>
                      <value>3a3a</value>
                   </resourcerequestparam>
                </requestParams>
                <requesterDn>cn=uaadmin,ou=sa,o=data</requesterDn>

<resourceDn>cn=Vodacom,cn=ResourceDefs,cn=RoleConfig,cn=AppConfig,cn=U
ser
Application Driver,cn=driverset1,o=system</resourceDn>
 </resourceassignment>
```

- *correlationId* specifies a resource assignment request correlation ID; if the parameter is null, a correlation ID is generated.

setResourceLocalizedStrings

Sets the localized strings for a resource, such as the names and descriptions.

A correlation ID is generated automatically for this method that uses this format:

```
UserApp#RemoteResourceRequest#xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Syntax: Here is the method signature:

```
public LocalizedValue[] setResourceLocalizedStrings(String resourceDn,
LocalizedValue[] locStrings, int type)
            throws NrfServiceException, RemoteException;
```

The parameters are described below:

- *resourceDn* specifies the DN of the resource for which you want to set the localized strings.
- *locStrings* provides an array of localized strings you want to define.
- *type* specifies the type of localized strings you want to retrieve. A type value of 1 retrieves a list of names for the resource, whereas a type value of 2 retrieves a list of descriptions.

setResourceLocalizedStringsAid

Sets the localized strings for a resource, such as the names and descriptions, with a correlation ID that you provide. The correlation ID is used for auditing to link a set of related resources.

Syntax: Here is the method signature:

```
public LocalizedValue[] setResourceLocalizedStringsAid(String resourceDn,
LocalizedValue[] locStrings, int type, String correlationId)
            throws NrfServiceException, RemoteException;
```

CodeMapRefreshStatus

Supporting class that provides details about the status of a code map refresh.

getConnectionName

Returns the name of the connected system.

Syntax: Here is the method signature:

```
public String getConnectionName()
```

getEntitlementDN

Returns the DN for the entitlement.

Syntax: Here is the method signature:

```
public String getEntitlementDN()
```

getGuid

Returns the GUID for the entitlement.

Syntax: Here is the method signature:

```
public String getGuid()
```

getLastRefresh

Returns the timestamp for the last refresh.

Syntax: Here is the method signature:

```
public long getLastRefresh()
```

getStatus

Returns the refresh status as a string indicating whether the refresh was successful.

Syntax: Here is the method signature:

```
public String getStatus()
```

setConnectionName

Sets the name of the connection system.

Syntax: Here is the method signature:

```
public void setConnectionName(final String connectionName)
```

setEntitlementDN

Sets the entitlement DN.

Syntax: Here is the method signature:

```
public void setEntitlementDN(String entitlementDN)
```

setGuid

Sets the GUID for the entitlement.

Syntax: Here is the method signature:

```
public void setGuid(String guid)
```

setLastRefresh

Sets the last refresh timestamp.

Syntax: Here is the method signature:

```
public void setLastRefresh(final long lastRefresh)
```

setStatus

Sets the refresh status.

Syntax: Here is the method signature:

```
public void setStatus(String status)
```

CodeMapValueStatus

Supporting class that provides details about the status of a refresh for a code map value.

getUpToDate

Returns true or false to indicate whether the status is up-to-date.

Syntax: Here is the method signature:

```
public boolean getUpToDate()
```

getRefreshStatus

Returns the refresh status as a CodeMapRefreshStatus object.

Syntax: Here is the method signature:

```
public CodeMapRefreshStatus getRefreshStatus()
```

getValue

Gets the code map value.

Syntax: Here is the method signature:

```
public String getValue()
```

setRefreshStatus

Sets the refresh as a CodeMapRefreshStatus object.

Syntax: Here is the method signature:

```
public void setRefreshStatus(final CodeMapRefreshStatus refreshStatus)
```

setUpToDate

Sets a boolean indicating whether the status is up-to-date.

Syntax: Here is the method signature:

```
public void setUpToDate(final boolean upToDate)
```

setValue

Sets the code map value.

Syntax: Here is the method signature:

```
public void setValue(final String value)
```

EntitlementRefreshInfo

Supporting class that provides refresh information for an entitlement after a code map refresh has been performed.

getDetailedStatus

Returns the detailed status as an array of CodeMapRefreshStatus objects.

Syntax: Here is the method signature:

```
public CodeMapRefreshStatus[] getDetailedStatus()
```

getEntitlementDN

Returns the DN for the entitlement.

Syntax: Here is the method signature:

```
public String getEntitlementDN()
```

getGuid

Returns the GUID for the entitlement.

Syntax: Here is the method signature:

```
public String getGuid()
```

getStatus

Returns the status of the refresh as a boolean flag.

Syntax: Here is the method signature:

```
public boolean getStatus()
```

setDetailedStatus

Sets the detailed status as an array of CodeMapRefreshStatus objects.

Syntax: Here is the method signature:

```
public void setDetailedStatus(final CodeMapRefreshStatus[] detailedStatus)
```

setEntitlementDN

Sets the DN for the entitlement.

Syntax: Here is the method signature:

```
public void setEntitlementDN(String entitlementDN)
```

setGuid

Sets the GUID for the entitlement.

Syntax: Here is the method signature:

```
public void setGuid(String m_guid)
```

setStatus

Sets the status as a boolean flag.

Syntax: Here is the method signature:

```
public void setStatus(boolean m_status)
```

ProvisioningCodeMap

Value class to hold code map information from the code map and code map label tables.

getDescription

Returns the description

```
public String getDescription()
```

getName

Returns the name.

```
public String getName()
```

getEntityKey

Returns the entity key.

```
public String getEntityKey()
```

getEntityType

Returns the entity type.

```
public int getEntityType()
```

getQueryKey

Returns the query key.

```
public String getQueryKey()
```

getViewId

Returns the view ID.

```
public String getViewId()
```

getLastRefreshed

Returns the timestamp for the last refresh.

```
public long getLastRefreshed()
```

setDescription

Sets the description.

```
public void setDescription(String description)
```

setName

Sets the name.

```
public void setName(String name)
```

setEntityKey

Sets the entity key.

```
public void setEntityKey(String entityKey)
```

setEntityType

Sets the entity type.

```
public void setEntityType(int entityType)
```

setQueryKey

Sets the query key.

```
public void setQueryKey(String queryKey)
```

setViewId

Sets the view ID.

```
public void setViewId(String viewId)
```

setLastRefreshed

Sets the timestamp for the last refresh.

```
public void setLastRefreshed(long lastRefreshed)
```

getLabels

Returns the code map labels.

```
public ProvisioningCodeMapLabel[] getLabels()
```

setLabels

Sets the code map labels.

```
public void setLabels(ProvisioningCodeMapLabel[] labels)
```

getEntitlementDn

Returns the DN for the entitlement.

```
public String getEntitlementDn()
```

setEntitlementDn

Sets the DN for the entitlement.

```
public void setEntitlementDn(String entitlementDn)
```

getDriverDn

Returns the DN for the driver.

```
public String getDriverDn()
```

setDriverDn

Sets the DN for the driver.

```
public void setDriverDn(String driverDn)
```

getDriverDisplayName

Returns the display name for the driver.

```
public String getDriverDisplayName()
```

setDriverDisplayName

Sets the display name for the driver.

```
public void setDriverDisplayName(String driverDisplayName)
```

Resource

Supporting class that provides information about resources.

getName

Returns the name of the resource.

```
public String getName()
```

setName

Sets the name of the resource.

```
public void setName(String name)
```

getDescription

Returns the description of the resource.

```
public String getDescription()
```

setDescription

Sets the description of the resource.

```
public void setDescription(String description)
```

getEntityKey

Returns the entity key for the resource.

```
public String getEntityKey()
```

setEntityKey

Sets the entity key for the resource.

```
public void setEntityKey(String entityKey)
```

getResourceCategoryKeys

Returns the keys for the resource categories.

```
public CategoryKey[] getResourceCategoryKeys()
```

setResourceCategoryKeys

Sets the keys for the resource categories.

```
public void setResourceCategoryKeys(CategoryKey[] resourceCategoryKeys)
```

getEntitlementRef

Returns the entitlement reference for the resource.

```
public NrfEntitlementRef[] getEntitlementRef()
```

setEntitlementRef

Sets the entitlement reference for the resource.

```
public void setEntitlementRef(NrfEntitlementRef[] entitlementRef)
```

getGrantApprovers

Returns the list of approvers for resource grant operations.

```
public Approver[] getGrantApprovers()
```

setGrantApprovers

Sets the list of approvers for resource grant operations.

```
public void setGrantApprovers(Approver[] grantApprovers)
```

getGrantQuorum

Returns the quorum condition for grant operations.

```
public String getGrantQuorum()
```

setGrantQuorum

Sets the quorum condition for grant operations.

```
public void setGrantQuorum(String grantQuorum)
```

getGrantRequestDef

Returns the provisioning request definition for grant operations.

```
public String getGrantRequestDef()
```

setGrantRequestDef

Sets the provisioning request definition for grant operations.

```
public void setGrantRequestDef(String grantRequestDef)
```

getRevokeQuorom

Returns the quorum condition for revoke operations.

```
public String getRevokeQuorum()
```

setRevokeQuorom

Sets the quorum condition for revoke operations.

```
public void setRevokeQuorum(String revokeQuorum)
```

getRevokeRequestDef

Returns the provisioning request definition for revoke operations.

```
public String getRevokeRequestDef()
```

setRevokeRequestDef

Sets the provisioning request definition for revoke operations.

```
public void setRevokeRequestDef(String revokeRequestDef)
```

getRevokeApprovers

Returns the list of approvers for revoke operations.

```
public Approver[] getRevokeApprovers()
```

setRevokeApprovers

Sets the list of approvers for revoke operations.

```
public void setRevokeApprovers(Approver[] revokeApprovers)
```

getOwners

Returns the list of owners for the resource.

```
public DNString[] getOwners()
```

setOwners

Sets the list of owners for the resource.

```
public void setOwners(DNString[] owners)
```

getParameters

Returns the list of entitlement parameters defined for the resource.

```
public ResourceParameter[] getParameters()
```

setParameters

Sets the list of entitlement parameters for the resource.

```
public void setParameters(ResourceParameter[] parameters)
```

getActive

Returns a boolean flag indicating whether the resource is still active, or has been approved or denied.

```
public boolean getActive()
```

setActive

Sets the boolean flag indicating whether the resource is still active.

```
public void setActive(final boolean active)
```

getAllowOverride

Returns a boolean flag indicating whether the approval process for the resource can be overridden by the approval process for a role.

```
public boolean getAllowOverride()
```

setAllowOverride

Sets the boolean flag indicating whether the approval process for the resource can be overridden by the approval process for a role.

```
public void setAllowOverride(final boolean allowOverride)
```

getAllowMulty

Returns a boolean indicating whether the resource allows a user to request multiple resource values.

```
public boolean getAllowedMulty()
```

setAllowMulty

Sets the boolean indicating whether the resource allows a user to request multiple resource values.

```
public void setAllowedMulty(final boolean allowedMulty)
```

ResourceAssignment

Supporting class that holds resource assignment information.

setResourceDn

Sets the DN for the resource.

```
public void setResourceDn(String resourceDn)
```

getResourceDn

Returns the DN for the resource.

```
public String getResourceDn()
```

setRequesterDn

Sets the DN for the requester.

```
public void setRequesterDn(String requesterDn)
```

getRequesterDn

Returns the DN for the requester.

```
public String getRequesterDn()
```

getRecipientDn

Returns the DN for the recipient of the assignment.

```
public String getRecipientDn()
```

setRecipientDn

Sets the DN for the recipient of the assignment.

```
public void setRecipientDn(String recipientDn)
```

getReason

Returns the reason for the assignment.

```
public String getReason()
```

setReason

Sets the reason for the assignment.

```
public void setReason(String reason)
```

getRequestDate

Returns the date of the assignment request.

```
public Date getRequestDate()
```

setRequestDate

Sets the date of the assignment request.

```
public void setRequestDate(Date requestDate)
```

setRequestParams

Sets the parameters for the request.

```
public void setRequestParams(ResourceRequestParam[] params)
```

getRequestParams

Returns the parameters for the request.

```
public ResourceRequestParam[] getRequestParams()
```

setInstanceGuid

Sets the instanceGuid for the resource assignment.

```
public void setInstanceGuid(String instanceGuid)
```

getInstanceGuid

Returns the instanceGuid for the resource assignment.

```
public String getInstanceGuid()
```

ResourceRequestParam

Supporting class that holds the name and value for a resource request parameter value.

ResourceRequestParam Constructors

The ResourceRequestParam class has two constructors.

Syntax 1: Here is the syntax for a constructor that takes no parameters:

```
public ResourceRequestParam()
    {
    }
```

Syntax 2: Here is the syntax for a constructor that takes two String parameters:

```
public ResourceRequestParam(String name, String value)
    {
        m_name = name;
        m_value = value;
    }
```

setName

Sets a parameter name.

Syntax: Here is the method signature:

```
public void setName(String name)
```

getName

Returns a parameter name.

Syntax: Here is the method signature:

```
public String getName()
```

setValue

Sets the value of a parameter.

Syntax: Here is the method signature:

```
public void setValue(String value)
```

getValue

Returns the value of a parameter.

Syntax: Here is the method signature:

```
public String getValue()
```

ResourceAssignmentRequestStatus

Supporting class that holds a resource request status item. The interface includes methods for getting and setting various request status properties. However, you will not need to call the methods for setting property values, since you are using this class to retrieve information about the request status. After calling the requestResourceGrant() or the requestResourceRevoke() methods, you can use the get methods to get the properties for each status object returned in the ResourceAssignmentRequestStatus array.

setEntityKey

Sets the entity key.

Syntax: Here is the method signature:

```
public void setEntityKey(String entityKey)
```

getEntityKey

Gets the entity key.

Syntax: Here is the method signature:

```
public String getEntityKey()
```

setReason

Sets the reason for the role assignment.

Syntax: Here is the method signature:

```
public void setReason(String reason)
```

getReason

Gets the reason for the role assignment.

Syntax: Here is the method signature:

```
public String getReason()
```

setStatusValue

Sets the status value for the request.

Syntax: Here is the method signature:

```
public void setStatusValue(int value)
```

setStatusDescription

Sets the status description for the request.

Syntax: Here is the method signature:

```
public void setStatusDescription(String description)
```

getStatusValue

Gets the status value for the request.

Syntax: Here is the method signature:

```
public int getStatusValue()
```

getStatusDescription

Gets the localized description for the request.

Syntax: Here is the method signature:

```
public String getStatusDescription()
```

setCorrelationId

Sets the correlation ID.

Syntax: Here is the method signature:

```
public void setCorrelationId(String correlationId)
```

getCorrelationId

Gets the correlation ID.

Syntax: Here is the method signature:

```
public String getCorrelationId()
```

setRequester

Sets the requester DN.

Syntax: Here is the method signature:

```
public void setRequester(String requester)
```

getRequester

Gets the requester DN.

Syntax: Here is the method signature:

```
public String getRequester()
```

setRequestDate

Sets the request date.

Syntax: Here is the method signature:

```
public void setRequestDate(Date requestDate)
```

getRequestDate

Gets the request date.

Syntax: Here is the method signature:

```
public Date getRequestDate()
```

setSource

Sets the source resource DN.

Syntax: Here is the method signature:

```
public void setSource(String source)
```

getSource

Gets the source resource DN.

Syntax: Here is the method signature:

```
public String getSource()
```

setTarget

Sets the DN for the target identity.

Syntax: Here is the method signature:

```
public void setTarget(String target)
```

getTarget

Gets the DN for the target identity.

Syntax: Here is the method signature:

```
public String getTarget()
```

setRequestParams

Sets the dynamic request parameters.

Syntax: Here is the method signature:

```
public void setRequestParams(ResourceRequestParam[] params)
```

getRequestParams

Gets the dynamic request parameters.

Syntax: Here is the method signature:

```
public ResourceRequestParam[] getRequestParams()
```

Resource Web Service Examples

This section provides examples of using the Resource Web Service.

Code Map Synchronization Code Samples

This section provides code samples for using the SOAP endpoints for code map synchronization.

```
public IRemoteResource stub;
stub=getResourcesStub(url,adminname,password);


//refreshCodeMap
EntitlementRefreshInfo refreshResult =
stub.refreshCodeMap("cn=Devices,cn=DevicesLoopback,cn=driverset1,o=system"
);
System.out.println(refreshResult .getDetailedStatus());
System.out.println(refreshResult .getEntitlementDN());
System.out.println(refreshResult .getGuid());
System.out.println(refreshResult .getStatus());


//getRefreshStatus
CodeMapRefreshStatus[] refreshStatus
=stub.getRefreshStatus("cn=Devices,cn=DevicesLoopback,cn=driverset1,o=syst
em");
for (CodeMapRefreshStatus item : refreshStatus) {
          System.out.println("Connection Name is: " +
item.getConnectionName());
              System.out.println("Entitlement DN is: " +
item.getEntitlementDN());
             System.out.println("Entitlement GUID is: " + item.getGuid());
             System.out.println("Last Refresh of this Entitlement is: " +
item.getLastRefresh());
              System.out.println("Status is: " + item.getStatus());
          }
```

```
//checkCodeMapValueStatus
String connectionName="SAP123";
CodeMapValueStatus checkStatus =
String codeMapValue=null;

stub.checkCodeMapValueStatus("cn=Devices,cn=DevicesLoopback,cn=driverset1,
o=system",connectionName, codeMapValue);

     System.out.println("Connection Name is: " +
checkStatus.getRefreshStatus().getConnectionName());
            System.out.println("Entitlement DN is: " +
checkStatus.getRefreshStatus().getEntitlementDN());
            System.out.println("Entitlement GUID is: " +
checkStatus.getRefreshStatus().getGuid());
            System.out.println("Last Refresh of this Entitlement is: " +
checkStatus.getRefreshStatus().getLastRefresh());
            System.out.println("Status is: " +
checkStatus.getRefreshStatus().getStatus());

            System.out.println(checkStatus.getUpToDate());
            System.out.println(checkStatus.getValue());




private static IRemoteResource getResourcesStub(String url,
    String username, String password) throws ServiceException {
         Stub stub = null;


         ResourceService service = new ResourceServiceImpl();
         stub = (Stub) service.getIRemoteResourcePort();
         stub._setProperty(Stub.USERNAME_PROPERTY, username);
         stub._setProperty(Stub.PASSWORD_PROPERTY, password);

         stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY,url +"/resource/
service");
     stub._setProperty(Stub.SESSION_MAINTAIN_PROPERTY, Boolean.TRUE);

         return (IRemoteResource) stub;
 }
```

31 Forgot Password Web Service

This section describes the Forgot Password Web Service, which allows SOAP clients to invoke a subset of the actions available through the Password Management system.

About the Forgot Password Web Service

The Forgot Password Web Service exposes a small set of actions from the Password Management system. The service allows remote clients to retrieve information about the forgot password configuration. In addition, it allows clients to retrieve information about the forgot password settings for a particular user, and perform challenge response and change password operations for a user.

The Forgot Password Web Service does not support the full range of password self-service operations. The Forgot Password Web Service is only for forgot password operations. If you want to create a custom user interface for performing password self service functions, such as answering or updating the user's hint or answer, or updating the challenge response questions, or checking on the password policy status, you need to use the REST endpoints that have been added to RBPM.

Calls to the Forgot Password Web Service require HTTP authentication.

Accessing the Service

You can access the Forgot Password Web Service endpoint using a URL similar to the following:

`http://server:port/warcontext/pwdmgt/service`

For example, if your server is named "myserver", your User Application is listening on port 8080, and your User Application war file is named "IDMPROV", the URL would be:

`http://myserver:8080/IDMPROV/pwdmgmt/service`

NOTE: The URL for the Forgot Password Web Service can be changed on the **Forgot Password Settings** page on the Administration tab in the User Application. To change the URL, enter the new URL in the **Forgot Password Web Service URL** field at the bottom of the page.

Accessing the WSDL

You can access the WSDL for the Forgot Password Web Service using a URL similar to the following:

`http://server:port/warcontext/pwdmgt/service?wsdl`

For example, if your server is named "myserver", your User Application is listening on port 8080, and your User Application war file is named "IDMPROV", the URL would be:

`http://myserver:8080/IDMPROV/pwdmgt/service?wsdl`

Generating the Stub Classes

Before using the Web Service, you need to use the WSSDK tool or another SOAP tool kit to generate the stub classes. To allow your code to find the stub classes, you also need to add the JAR that contains the stub classes to your classpath.

If you want to use the NetIQ WSSDK tool, you can generate the client stubs by extracting the WSDL and running the wsdl2java utility. For example, you could run this command to generate the stubs in a package called com.novell.soa.af.pwdmgt.soap.impl:

```
"C:\Program Files\Java\jdk1.6.0_31\bin\java" -cp "../lib/wssdk.jar;../lib/
jaxrpc-api.jar";"../lib/mail.jar";"../lib/activation.jar";"c:\Program
Files\Java\jdk1.6.0_31\lib\tools.jar";
com.novell.soa.ws.impl.tools.wsdl2java.Main -verbose -ds gensrc -d C:\ -
noskel -notie -genclient -keep -package com.novell.soa.af.pwdmgt.soap.impl
-javadoc pwdmgt.wsdl
```

You can change the wsdl2java parameters to suit your requirements.

Password Management Web Service Interface

This section provides reference information for each forgot password operation available through the Password Management interface.

processForgotConf

Gets the forgot password configuration parameters.

This method returns an object of type ForgotPasswordConfWSBean. This object contains the following information about the configuration:

Table 31-1 *ForgotPasswordConfWSBean Data*

Field	Description
Configured Return Link	Provides the forgot password return link.
Show Return Link	Indicates whether to show the forgot password return link.

Syntax: Here is the method signature:

```
public ForgotPasswordConfWSBean processForgotConf()
            throws RemoteException;
```

processUser

Retrieves forgot password configuration information for a user.

This method returns an object of type ForgotPasswordWSBean. If no match is found for the the user name specified, an error message is returned in the getUsers() method of ForgotPasswordWSBean. If multiple matches are found, the getUsers() method is returned with a String array of users. If a single match is found, the getUsers() method has a length of 1, and the following methods in ForgotPasswordWSBean are set:

- getConfiguredRtnLink()
- getShowReturnLink()
- getShowHint()
- getHint()
- getShowFullDN()
- getUserDisplayDN()
- getUserDN()
- getUser()
- getMessage()
- getAction()
- getChallengeQuestions()
- getChaResInUser()
- getMessage()

When a single user match is found, the user should be presented with the Challenge Response screen. If getChaResInUse() returns false, then call processChaRes() and show the Forgot Success screen directly without presenting the Challenge Response screen.

Syntax: Here is the method signature:

```
public ForgotPasswordWSBean processUser(final String userName)
            throws RemoteException;
```

The parameters are described below:

- *userName* specifies the name of a user.

processChaRes

Processes one or more challenge response answers for a particular user.

If the challenge response operation is authenticated, the following events may occur:

- If the password policy action is EmailHint, the operation will send an email with the hint to the user, and set the message to indicate that the operation succeeded. Therefore, the caller of this method should go to the Forgot Password Change Success screen, and display the message.
- If the password policy action is ShowHint, the operation will set the message to the user's hint. Therefore, the caller of this method should go to the Forgot Password Change Success screen, and display the message with the hint on the page.

- If the password policy action is EmailPassword, the operation will set send the password to the user. Therefore, the caller of this method should go to the Forgot Password Change Success screen, and display the message.
- If the password policy action is ChangePassword, the operation will set the password rules and the password hint. Therefore, the caller of this method should go to the Forgot Password Change screen.

This method returns an object of type ForgotPasswordWSBean. After the processCharRes operation is called, the following methods are populated with values:

- getTimeout()
- getRules()
- getLocked()
- getError()
- getMessage()

If the getAction() method returned by the processUser() operation is ChangePassword, then present the user with the Password Change screen. Otherwise, go to the Forgot Success screen and present the user with the message returned from the getMessage() method.

Syntax: Here is the method signature:

```
public ForgotPasswordWSBean processChaRes(final String userDN, final
String[] chaAnswers) throws RemoteException;
```

The parameters are described below:

- *userDN* specifies the DN for a particular user.
- *chaAnswers* provides an array of challenge response answers. The answers are processed in the order in which they are presented.

processChgPwd

Resets the password for a particular user.

After the processChgPwd operation is called, the following events may occur:

- If the change password operation succeeds, the caller of this method should go to the Forgot Password Success screen, and display the success message.
- If the change password operation fails, the error field on the ForgotPasswordWSBean object is set to true, and the message field is populated with the corresponding error message. Therefore, the caller of this method should stay on the password screen and display the error message.

This method returns an object of type ForgotPasswordWSBean. After the processChgPwd operation is called, the following methods are populated with values:

- getTimeout()
- getError()

If the getError() method returns false, you need to present the user with the Password Change Success screen.

Syntax: Here is the method signature:

```
public ForgotPasswordWSBean processChgPwd(final String userDN, final String
newPassword, final String confirmPassword )
            throws RemoteException;
```

The parameters are described below:

- *userDN* specifies the DN for a particular user.
- *newPassword* supplies a password for the user.
- *confirmPassword* repeats the password for confirmation.

ForgotPasswordWSBean

Here is the complete structure of the ForgotPasswordWSBean object:

Table 31-2 *ForgotPasswordWSBean Structure*

Field	Description
Users	Provides a list of the users that match the search criteria specified. When the wildcard feature is enabled, multiple matches may be found.
Challenge Questions	Supplies the challenge questions associated with the user.
Configured Return Link	Shows the Return link to be used after the user performs a forgot password operation.
Show Return Link	Indicates whether to show the Return link after the user performs a forgot password operation.
Show Hint	Indicates whether to show the user's password hint on the Forgot Password Change screen.
Show Full DN	Indicates whether to show the user's full DN or just the CN name after the user performs a forgot password operation.
User DN	Shows the user's DN.
User Display DN	Shows the user's display DN. For example, `cn=ablake,ou=users,o=netiq` or `workforceID=ablake,ou=users,o=netiq`.
User	Provides the user's display name.
Error	Returns true if an error occurs.
Message	Returns a message in the event that there is an application-specific error.
Action	Specifies the policy action, which is one of the following values: ShowHint, EmailHint, EmailPassword, ChangePassword.

Field	Description
Hint	Specifies the user's password hint.
Rules	Lists the password policy rules.
Is Challenge Response in User	Indicates whether the challenge response feature is enabled for this user. If challenge response in use is false, then the user can only perform the email hint and show hint functions.
Locked	Indicates whether the user account is locked.
Timeout	Indicates whether a session timeout occurred.
Login Attribute	Specifies the user's Login Attribute.

VI Configuring Single Sign-on Access in Identity Manager

By default, Identity Manager uses OSP for single sign-on access in Identity Manager. When you install Identity Reporting and the identity applications, you specify the basic settings for user authentication. However, you can also configure the OSP authentication server to accept authentication from the Kerberos ticket server or SAML. For example, you can use SAML to support authentication from NetIQ Access Manager. You can also enable single-sign by configuring Access Gateway as a reverse proxy server that provides single sign-on and restricts access to the identity applications and Identity Reporting servers by securely providing credential information for authenticated users. For more information about OSP, "Understanding How OSP Works with Identity Manager" on page 538.

32 Preparing for Single Sign-on Access

NetIQ recommends that you complete the steps in the following checklist:

	Checklist Items
❑	1. Understand how OSP works in Identity Manager. For more information, see "Understanding How OSP Works with Identity Manager" on page 538.
❑	2. Understand how Identity Manager uses OSP for single sign-on access. For more information, see "Understanding Authentication with One SSO Provider" on page 534.
❑	3. Install the identity applications and password management components. For more information, see the installation guide for your platform.
❑	4. (Optional) Install Identity Reporting. For more information, see the installation guide for your platform.
❑	5. Configure the identity applications for single sign-on access using OSP. For more information, see "Using One SSO Provider for Single Sign-on Access in Identity Manager" on page 543.
❑	6. Install the authentication system that you want to use with Identity Manager. For example, Access Manager or Kerberos.
❑	7. (Conditional) Configure Access Manager and OSP. For more information, see the following: ◆ "Using SAML Authentication for Single Sign-on" on page 546 ◆ "Reverse Proxy Based Single Sign-On" on page 549
❑	8. Verify the single sign-on settings. For more information, see Chapter 38, "Verifying Single Sign-on Access for the Identity Applications," on page 579.

33 Using Self-Service Password Management in Identity Manager

Identity Manager includes NetIQ Self Service Password Reset (SSPR) to help users who have access to the identity applications to reset their passwords without administrative intervention. The installation process enables SSPR by default when you install or upgrade to the latest version of Identity Manager. In a new installation, SSPR uses a proprietary protocol for managing authentication methods. However, after an upgrade, you can instruct SSPR to use the NetIQ Modular Authentication Services (NMAS) that Identity Manager traditionally has used for its legacy password management program.

Depending on whether you want to use complex password management, you can configure one of the following providers:

SSPR

NetIQ Self Service Password Reset is the default option when you install or upgrade Identity Manager. For more information, see "Understanding the Default Self-Service Process" on page 533.

Legacy Provider for Password Management

Uses the password management process from Identity Manager 4.0.2, which supports the use of multiple password policies. For more information, see "Understanding the Legacy Password Management Provider" on page 534.

Third-Party Provider Password Management

You can use an third-party program for managing forgotten passwords. You need to modify some configuration settings for Identity Manager. For more information, see Using an External System for Forgotten Password Management in the *NetIQ Identity Manager Setup Guide for Windows*.

Understanding the Default Self-Service Process

SSPR automatically integrates with the single sign-on process for the identity applications and Identity Reporting. It is the default password management program for Identity Manager, even when you do not install SSPR. When a user requests a password reset, SSPR requires the user to answer the challenge-response question. If the answers are correct, SSPR responds in one of the following ways:

- Allow users to create a new password
- Create a new password and send it to the user
- Create a new password, send it to the user, and mark the old password as expired.

You configure this response in the SSPR Configuration Editor. After upgrading to a new version of Identity Manager, you can configure SSPR to use the NMAS method that Identity Manager traditionally has used for password management. However, SSPR does not recognize your existing password policies for managing forgotten passwords. To continue using your policies, see "Understanding the Legacy Password Management Provider" on page 534.

You also can configure SSPR to use its proprietary protocol instead of NMAS. If you make this change, you cannot return to using NMAS without resetting your password policies.

Understanding the Legacy Password Management Provider

NOTE: The Legacy Password Self-Service feature of the User Application is deprecated with this release. NetIQ strongly recommends that you start using SSPR for all password-specific tasks. The installation process enables SSPR by default.

When you upgrade from an older version of Identity Manager, the identity applications default to SSPR as the password management program. SSPR can use the NMAS method that Identity Manager traditionally has used for password management. However, SSPR does not recognize your existing password policies for managing forgotten passwords. You can bypass SSPR and use the legacy password management provider.

When a user requests a password reset, the legacy provider compares the user's credentials to the password policies that you set. For example, it might requires the user to answer a challenge-response question. Based on the policy applied to that user, the program responds in one of the following ways:

- Resets the password
- Shows the password hint
- Emails the password hint to the user
- Emails a new password to the user

Use the legacy provider if your enterprise uses multiple or complex password policies. For example, your password policies are based on user roles. An intern might simply need a auto-generated password without a challenge response. For a manager who can access secure data, you might have more stringent requirements. This user might need to regularly reset the password. In both cases, you want the users to have self-service for password requests.

To use the legacy provider, modify the configuration settings for the identity applications after you install or upgrade Identity Manager. You do not need to reconfigure your password policies after the upgrade.

Understanding Authentication with One SSO Provider

OSP supports the OAuth2 specification and requires an LDAP authentication server. By default, Identity Manager uses Identity Vault (eDirectory). OSP can communicate other types of **authentication sources**, or **identity vaults**, to handle the authentication requests. You can configure the type of authentication that you want OSP to use: userID and password, Kerberos, or SAML. However, OSP does not support MIT-style Kerberos or SAP login tickets.

How do OSP and SSO work?

If you use the Identity Vault as your authentication service and the specified containers in the Identity Vault have CNs and passwords, authorized users can log in to Identity Manager immediately after installation. Without these login accounts, only the administrator that you specify during installation can log in immediately.

When a user logs in to one of the browser-based components, the process redirects the user's name/password pair to the OSP service, which queries the authentication server. The server validates the user credentials. Then OSP issues an OAuth2 access token to the component and browser. The browser uses the token during the user's session to provide SSO access to any of the browser-based components.

If you use Kerberos or SAML, OSP accepts authentication from the Kerberos ticket server or SAML IDP then issues an OAuth2 access token to the component where the user logged in.

How does OSP work with Kerberos?

OSP and Kerberos ensure that users can log in once to create a session with one of the identity applications and Identity Reporting. If the user's session times out, authorization occurs automatically and without user intervention. After logging out, users should always close the browser to ensure that their sessions end. Otherwise, the application redirects the user to the login window and OSP reauthorizes the user session.

How do I set up Authentication and Single Sign-on Access?

For OSP and SSO to function, you must install OSP. Then specify the URLs for client access to each component, the URL that redirects validation requests to OSP, and settings for the authentication server. You can provide this information during installation or afterward with the RBPM configuration utility. You can also specify the settings for your Kerberos ticket server or SAML IDP.

How OSP Works with Identity Manager

The identity applications provide authentication and single sign-on (SSO) through the One SSO Provider service (OSP). OSP authenticates a user or a service on behalf of identity applications. For OSP to function, you must install OSP included in the Identity Manager installation package. The identity applications set up a trust relationship with an OSP instance via shared secrets and a public or private key pair through TLS protocol.

After authenticating an entity, OSP sends an access token to the identity applications based on OAuth 2.0 protocol. The identity applications use the token to obtain the identity information about the authenticated entity. The identity applications then use the identity information to perform authorization to resources controlled by the identity applications.

The identity applications components interact with OSP through OAuth 2.0, where OSP acts as the OAuth 2.0 provider. Although OSP supports a variety of authentication mechanisms such as username/password, Kerberos and SAML, this chapter explains OSP configuration through OAuth 2.0.

OSP supported with OAuth 2.0 specification requires an LDAP authentication server. By default, Identity Manager uses Identity Vault as an authentication server. OSP can communicate with other types of authentication sources, or identity vaults, to handle the authentication requests. You can configure the type of authentication that you want OSP to use, such as user ID and password, Kerberos, or SAML. If you use Kerberos or SAML, OSP accepts authentication from the Kerberos

ticket server or SAML Identity Provider (IDP) and then issues an OAuth 2.0 access token to the component where the user logged in. However, OSP does not support MIT-style Kerberos or SAP login tickets.

OSP APIs expose all OAuth functionalities as endpoints for obtaining access tokens. The identity applications expose these APIs. For example, OSP provides an HTTP endpoint that an application uses to validate a token and obtain the identity information associated with the token. OSP also provides an HTTP endpoint that allows an identity application to inform OSP that a user has requested to log out of the application.

OSP Concepts

This section describes the basic concepts of OSP.

OSP Configuration

The configuration template for OSP resides in the `osp-conf.jar` file. The template includes information from the following resources:

- `ism-configuration.properties` file
- User Application configuration locations including Identity Vault and the file system

Most of the OSP properties are configured by using the Configuration Update utility (`configupdate.sh` or `configupdate.bat`).

Access Tokens and Refresh Tokens

OAuth-defined authentication provides two types of tokens: **access token** and **refresh token**.

Access token is a bearer token. Any entity that obtains an access token may use the token. To keep away the chances of unauthorized access, the lifetime of an access token is typically very short.

Refresh token is used when possibility of token leakage is low. For example, when the token is held on a server than in a browser. A refresh token can be used to obtain an access token without requiring the user to re-enter the credentials. The lifetime of a refresh token is typically quite long.

Authentication Timeouts

The authentication process involves a number of different and related timeout values. The two most important timeout values are:

- OAuth access token lifetime (default value is 60 seconds)
- OSP session timeout (session time-to-live; default value is 45 minutes)

Each time an access token expires, the application requests a new access token from OSP. Each time a browser request is made to OSP, the session time-to-live token is reset. Therefore, as long as a user is using an application and the application is getting new access tokens, the user is not logged out.

A refresh token allows an application to obtain a new access token without user interaction. Refresh tokens are used by applications that can keep them secure. Therefore, refresh tokens have long lifetimes (default 30 days although effective lifetime is 48 hours due to revocation timeout). The identity applications use refresh tokens in the backend.

OSP automatically revokes a refresh token that was obtained through a browser-based request when an OSP session is logged out.

Applications that obtain refresh token through a backend request use the `http[s]://<host>[:port]/osp/a/idm/auth/oauth2/revoke` endpoint as described in RFC 7009. A refresh token obtained through OAuth 2.0 Resource Owner Password Grant (backend name/password method) must be manually revoked. If a refresh token is not revoked through session logout or through a backend request, then the token revocation information remains in the oidpInstanceData (osp.sch) attribute on the LDAP user object. When a user does not log out of the identity applications, OSP does not remove the login entry from the oidpInstanceData. If the user continues to log in without logging out, the size of the entry grows large and prevents OSP from updating the attribute and may cause a login failure for the user. For troubleshooting this issue, see "Managing the Size of oidPInstancedata Attribute" on page 624.

OSP Cookies

OSP uses the following types of cookies in the authentication process:

- **OSP Session Cookie:** Records user information, such as user identifier and time-to-live. It is stored in temporary memory and not retained after the browser is closed. This type of cookie is removed when a user closes the web browser. Absence or expiry of this cookie means that the user is not authenticated. An OSP session cookie is represented as `x-oidp-session<group of hex chars>`.
- **OAuth2 Cookie:** Records OAuth 2.0 state between redirects. The default settings of Microsoft browsers sometimes prevent the submission of OSP cookies to the OSP server that can be determined by using browser developer tools or the OSP log. An OAuth2 cookie is represented as `x-oidp-oauth2-<group of hex chars>`.

Browser Code for the OAuth 2.0 Interaction

The identity applications require JavaScript to implement interactions with OSP.

The open source project, Spiffy UI, provides an easy way to interact with OSP when using Google Web Toolkit (GWT). For detailed information, see the following resources:

- http://www.spiffyui.org
- https://github.com/spiffyui?tab=repositories

If using AngularJS, the following project may be helpful: https://github.com/Gromit-Soft

OSP URLs

OSP uses the following URLs in authentication:

- grant
- getattributes
- logout

Examples used in the following sections assume a secured connection (HTTPS), where OSP is hosted on a server named `authentication host` with the default port 8443, and the tenant is named `idm`.

grant URL

Invoke this URL in a browser: `https://authenticationhost:8443/osp/a/idm/auth/oauth2/grant`

This is the OAuth 2.0 Authorization endpoint defined by Section 3.1 of RFC 6749.

An implicit grant request can look similar to this:

```
https://authenticationhost:8443/osp/a/idm/auth/oauth2/
grant?response_type=token&redirect_uri=http://applicationhost:8180/
landing/com.netiq.ualanding.index/
oauth.html&client_id=ualanding&state=spiffystate0.5457210745662451
```

The allowable HTTP methods and parameters are as defined in Section 4 of RFC 6749.

getattributes URL

Invoke this URL in a browser: `https:// authenticationhost:8443/osp/a/idm/auth/oauth2/getattributes`

This is the OAuth 2.0 Token endpoint defined by Section 3.2 of RFC 6749. The identity applications issue this token to obtain identity information about the entity for which an access token was created. The identity information is presented in the form of attributes (name-value pairs).

A token validation request from a web page can look similar to this:

```
https://authenticationhost:8443/osp/a/idm/auth/oauth2/
getattributes?attributes=name+expiration&access_token=eHwA[...etc.]&callba
ck=jQuery20305550091890618205_1463435676921&_=1463435676922
```

The desired attributes are specified by a space-separated list of attribute names using the “attributes” query item. The available attributes and their names are primarily defined by the tenant's authentication configuration in authcfg.xml. There are also several pre-defined attributes related to token management.

logout URL

Invoke this URL in a browser: `https://authenticationhost:8443/osp/a/idm/auth/app/logout`

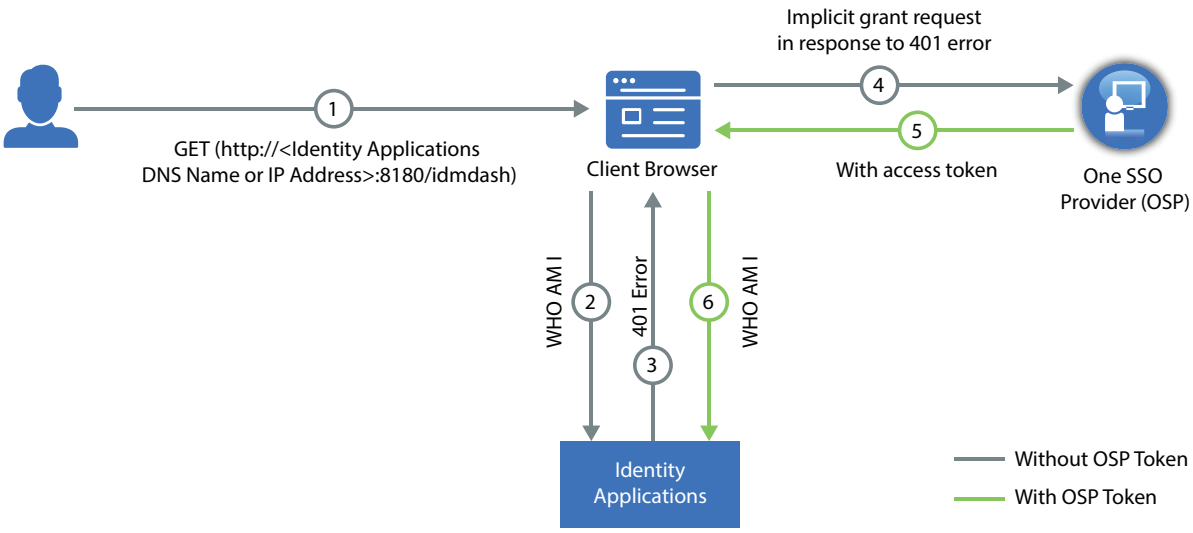
This is an OSP-specific endpoint that identity applications use to inform OSP that a user has requested a logout and that OSP must invalidate the OSP session information from the encrypted browser cookie. Typically this endpoint is invoked when a user selects **Logout** link on the identity application's web page.

Understanding How OSP Works with Identity Manager

When using OAuth 2.0, the Identity Manager communication with OSP primarily consists of the following actions:

1. Obtain an access token in the client browser.
2. Validate the access token when the client browser submits it to the identity applications server.

The following figure illustrates the components and process flow involved in basic authentication:



1 A user navigates to the identity application's home page.

2 The browser loads the identity application home page.

3 The application web page makes an HTTP request to the identity applications server endpoint.

   Note that the application has not yet obtained an authentication token. Therefore, it does not have any authentication information in the request and the identity applications server returns an HTTP 401 status (unauthorized).

4 The identity applications web page recognizes that 401 status means that the web page must obtain a valid authentication token. The web page directs the client browser to OSP `grant` endpoint for an implicit grant request. In the implicit flow, OSP issues the client browser an access token instead of an authorization code. For more information about implicit grant type, see Section 4.2 of RFC 6749.

   At this point, no authentication information exists because the OSP session cookie is either not present or, if present, indicates that the browser session is unauthenticated. Therefore, OSP presents a login web page.

5 The user enters authentication credentials (typically user name and password) on the OSP page and selects **Next**.

   OSP looks up the user in the Identity Vault and validates the user-entered password. You can also configure OSP for one or more additional authentication factors.

6 After the user is authenticated using the provided credentials, OSP creates an access token and redirects the browser to a `redirect URI` that the web application has provided. The access token and other data defined by OAuth 2.0 is appended to the redirect URI as a URI fragment. In addition, OSP returns its secure, encrypted session cookie to the browser.

7 The browser loads Identity Applications web page indicated by the redirect URI. Identity Applications then extracts the access token from the URI fragment and validates the state information contained in the fragment.

NOTE: The redirection endpoint URI must be an absolute URI as defined by Section 4.3 of RFC3986. The endpoint URI may include an `application/x-www-form-urlencoded` formatted query component according to Section 3.4 of RFC3986, which must be retained when adding additional query parameters. The endpoint URI must not include a fragment component. For more information, see sections 4.2.1 and 4.2.2 of RFC 6759.

---

8. The identity applications inject the access token in the HTTP Authorization header (typically using the `Bearer` authentication type) and request the identity applications server for validating the token.
9. The identity applications server contacts OSP via a back-channel HTTP request to validate the supplied token and to obtain identity information associated with the token. The identity applications server then responds to the HTTP request appropriately.
10. If the token is validated, the identity applications web pages allow the user to perform the tasks the user is allowed to do. When additional HTTP requests are made to the identity applications server, the access token is supplied as part of the request in an HTTP authorization header.
11. The access token in use is set to expire within the default expiration time interval of two minutes. If the identity applications server attempts to validate an access token after the token has expired, OSP informs the identity applications server that the token is no longer valid. The identity applications server then responds with an HTTP 401 status. The web application page again directs the client browser to OSP to request an access token.
12. OSP determines (via OSP's secure session cookie) that the user is still authorized (the session has not timed-out due to inactivity and has not been explicitly logged out) and redirects the user back to the web application page with an access token without asking the user for re-entering the credentials.

To understand the authentication process through REST endpoints, see "OSP Login Request Example by Using REST Endpoints" on page 622

Guidelines for Enabling OSP Logging

The OSP log level is controlled by `com.netiq.idm.osp.tenant.logging.level` property typically set in the `setenv.sh` file in Tomcat's `bin` directory. For example, `/opt/netiq/idm/apps/tomcat/bin/setenv.sh` on Linux. The `setenv.sh` file has the below entry at the end of the file.

```
JAVA_OPTS="-Xms1024m -Xmx1024m -XX:MaxPermSize=512m "
export JAVA_OPTS
export CATALINA_OPTS="-Dcom.netiq.ism.config=/opt/netiq/idm/apps/tomcat/
conf/ism-configuration.properties -Dcom.netiq.osp.ext-context-file=/opt/
netiq/idm/apps/osp_sspr/osp/osp-conf.jar -
Dcom.netiq.idm.osp.logging.level=INFO -
Dcom.netiq.idm.osp.client.host=myserver.acme.com -
Dcom.netiq.idm.osp.tenant.logging.naudit.enabled=false -
Dcom.netiq.idm.osp.logging.file.dir=${CATALINA_BASE}/logs -
Djava.awt.headless=true -Dfile.encoding=UTF-8 -Dsun.jnu.encoding=UTF-8 -
Djavax.xml.transform.TransformerFactory=org.apache.xalan.processor.Transfo
rmerFactoryImpl -Didmuserapp.logging.config.dir=/opt/netiq/idm/apps/
tomcat/conf/ -Dextend.local.config.dir=/opt/netiq/idm/apps/tomcat/conf/"
```

The property, `-Dcom.netiq.idm.osp.logging.level=ALL`, controls the amount of information that OSP logs. The Apache Foundation defines the following trace levels for log4j:

- `OFF`
- `FATAL`
- `ERROR`
- `WARN`

- INFO
- DEBUG
- TRACE
- ALL

By default, OSP logging is set to INFO. You can set other levels depending on what you are troubleshooting. After setting the log level, restart Identity Applications. For example, restart Tomcat by performing the following command:

```
systemctl restart netiq-tomcat.service
```

Before enabling logging, NetIQ recommends to review the following guidelines:

- Use ALL to troubleshoot if OSP is able to find the certificate that you included. This level names every single certificate in the known keystores it uses. This information can be useful because JVM has 90 or more certificates. In general, set the log level to ALL to debug or troubleshoot common issues. To generate additional messages, set com.netiq.idm.osp.debug property to true either as a Java system property in setenv.sh or ism-configuration.properties file.
- File logging is enabled by default. OSP creates files name as `osp-idm-<date of log generation>.log` file in the Tomcat directory. For example, `/opt/netiq/idm/apps/tomcat/logs/`. File logging records the actions that have occurred. For example, you can configure logging to list every request made to OSP. This can help you get a good idea of how often visitors are coming and how they navigate the application pages. The content logged to file logging can be controlled by specifying logger levels.
- When you enable console logging, OSP generates log messages in the `catalina.out` file located in the `logs` directory under Tomcat's root directory. For example, `/opt/netiq/idm/apps/tomcat/logs/catalina.out`. NetIQ recommends you to use file logging on Windows.
- OSP can handle thousands of requests per second. If transaction volume is high and each log entry consumes a few hundred bytes, OSP can fill up the available disk space in a matter of minutes. Logging also increases system overhead, which causes some degradation in system performance. Therefore, refrain from using console logging in a production environment because there is no default way to limit the size of the `catalina.out` file. For production use, the logging level should be set to WARNING or less. More verbose logging levels result in much more log data which requires both CPU resources to generate the log messages and disk resources to store the log messages.

34 Using One SSO Provider for Single Sign-on Access in Identity Manager

To provide single sign-on access to the identity applications, you must configure the settings in the RBPM Configuration utility. You should already have the certificates and keys necessary for single sign-on from installing OSP.

This procedure assumes that your environment will use one certificate for eDirectory, the SSO controller, and the OAuth Provider. If your organization requires additional layers of separation, create a separate certificate for the OAuth Provider.

Preparing eDirectory for Single Sign-on Access

You must configure the Identity Vault, as part of your eDirectory installation, to support single sign-on access for the identity applications and Identity Reporting.

The eDirectory Administrator should create value indexes for the manager, ismanager and srvprvUUID attributes. Without value indexes on these attributes, identity applications users can experience impeded performance, particularly in a clustered environment. You can create these value indexes automatically during installation by selecting Advanced > Create eDirectory Indexes in the RBPM Configuration utility. For more information about using Index Manager to create value indexes, see the *NetIQ eDirectory Administration Guide*.

If you previously extended the eDirectory schema to include the SAML schema and installed the required NMAS methods, you do not need to perform those steps a second time. Instead, skip to the subsection about creating the Trusted Root Container.

Modifying the Basic Settings for Single Sign-on Access

When you install the identity applications, you generally configure the basic settings for single sign-on access. This section helps you ensure that the settings work for your environment.

1. Run the RBPM Configuration utility.
2. To modify the authentication settings, complete the following steps:
   - **2a** Click **Authentication**.
   - **2b** (Conditional) To specify the actual server DNS name or IP address, change all instances of `localhost`.
     - The specified address must be resolvable from all clients. Use `localhost` only if all access to Identity Manager will be local, including access through a browser.

- This “public” host name or IP address should be the same as the value of *PublicServerName* that you specified when you installed OSP.
- In a distributed or clustered environment, all of the OAuth URLs should be the same value. The URL should drive client access through your L4 switch or load balancer. Also, the `osp.war` and configuration files must be installed on each deployment in the environment.

2c For **LDAP DN of Admins Container**, click the **Browse** button, then select the container within the Identity Vault that contains your identity applications administrator.

2d Specify the OAuth keystore file that you created when you installed OSP.

Include the keystore file path, keystore file password, key alias, and key password. The default keystore file is `osp.jks`, and the default key alias is `osp`.

3 To modify the single sign-on settings, complete the following steps:

3a Click **SSO Clients**.

3b (Conditional) To specify the actual server DNS name or IP address, change all instances of `localhost`.

- The specified address must be resolvable from all clients. Use `localhost` only if all access to the Dashboard will be local, including access through a browser.
- This “public” host name or IP address should be the same as the value of *PublicServerName* that you specified when you installed OSP.
- In a distributed or clustered environment, all of the OAuth redirect URLs should be the same value. The URL should drive client access through your L4 switch or load balancer.

3c (Conditional) If you use non-default ports, update the port numbers for the following Identity Manager components:

- Identity Applications Administration
- Identity Manager Dashboard
- Identity Reporting
- User Application

4 Click **OK** to save your changes, then close the configuration utility.

5 Start Tomcat.

Configuring Self Service Password Reset to Trust OSP

In order for single sign-on to work properly, you must configure a trust relationship with certificates between the OSP and Self Service Password Rest (SSPR). To establish a connection between OSP and SSPR, you must export the `osp` certificate from `osp.jks`.

After you export the certificate, you must import the certificate in to the keystore file for SSPR. The default path keystore file for SSPR is:

- **Linux/UNIX:** */[Java_Home]/lib/security/cacerts*
- **Windows:** `C:\[Java_Home]\lib\security\cacerts`

For more information about setting a secure channel, see “Setting Up a Secure Channel Between the Application Server and the LDAP Server ” in the “Self Service Password Reset Administration Guide”.

35 Using NetIQ Access Manager for Single Sign-On

This section helps you configure NetIQ Access Manager and OSP to support single sign-on access in Identity Manager.

Before beginning, review the following assumptions for these instructions:

- You have installed a new, supported version of Access Manager.
- You have installed a new version of Identity Manager.
- Both installations use DNS names for the host name configuration.
- Both installations use SSL protocol for communication.
- (Conditional) For reverse-proxy single sign-on service, Access Manager and Identity Manager are pointing to the same user store for authentication.
- (Conditional) For SAML supported authentication, Access Manager and Identity Manager are pointing to any supported user store. For more information, see *NetIQ Access Manager Administration Guide*.
- You have installed the certificate using DNS on application server where OSP is installed.

You need to be familiar with NetIQ Access Manager capabilities so that you understand the context of the content in this section. For more information about NetIQ Access Manager, see the Access Manager documentation website.

Understanding Third-Party Authentication and Single Sign-On

You can configure Identity Manager to work with NetIQ Access Manager using SAML 2.0 or by configuring Access Gateway as a reverse proxy server.

SAML 2.0

Enables you to use a technology that is not password-based to log in to the identity applications through Access Manager. For example, users can log in through a user (client) certificate, such as from a smart card.

Access Manager interacts with OSP to map the user to a DN in the Identity Vault. When a user logs in to the identity applications through Access Manager, Access Manager can inject a SAML assertion (with the user's DN as the identifier) into an HTTP header and forward the request to the identity applications. The identity applications use the SAML assertion to establish the LDAP connection with the Identity Vault.

Accessory portlets that allow single sign-on authentication based on passwords do not support single sign-on when SAML assertions are used for identity application authentication.

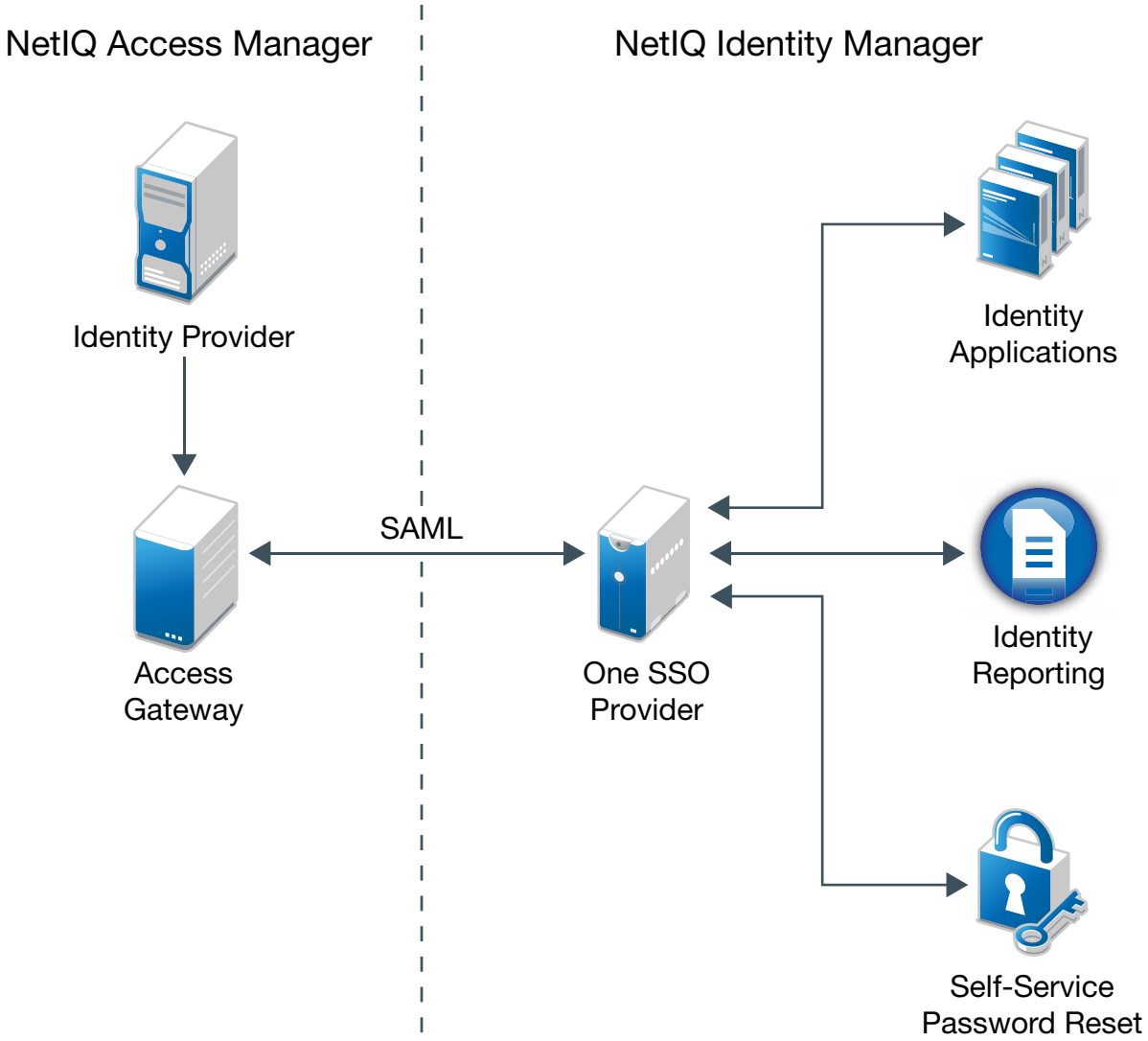
Reverse Proxy

Protects the identity applications by creating a reverse proxy that acts as the front end to your identity applications in your Identity Manager environment. In this approach, Access Gateway uses a Form Fill policy for single sign-on authentication to the identity applications. A reverse proxy can be configured to protect one or more proxy services by using a domain or path based proxy service, single sign-on access, and simultaneous logout.

The identity applications retrieve the access token from OSP and provide access to the user. This completes the single sign-on process for the first login. The identity applications uses this access token for providing single sign-on access for future authentication requests to any of the identity application and SSPR.

Using SAML Authentication for Single Sign-on

This section helps you configure both NetIQ Access Manager and OSP to support single sign-on access in Identity Manager using SAML 2.0 authentication.



Establishing Trust between Identity Manager and Access Manager

Identity Manager needs the URL of the SAML metadata to redirect users for authentication requests. By default, Access Manager uses the following URL for storing the SAML metadata:

`https://server:port/nidp/saml2/metadata`

where *server*:*port* represent the Access Manager Identity Server.

1. (Optional) To view an `.xml` document for the SAML metadata, open the URL in a browser.

   If the URL does not produce the document, ensure that the link is correct.
2. Launch configupdate utility on the OSP server.
3. Click **Advance** to view more options.
4. Select **Authentication**.
5. In the **Authentication Server** section, specify the DNS name of the server that hosts OSP in the **Oauth server host identifier** setting.
6. For **Authentication Method**:
   - **6a** Select **SAML 2.0** from the **Method** list.
   - **6b** Select **URL** from the **Metadata source** list.
   - **6c** In **Metadata URL**, specify the URL that OSP uses to redirect the authentication request to SAML metadata of Access Manager.

     For example, `https://server:port/nidp/saml2/metadata`
   - **6d** Select **Load on exit** and **Configure Access Manager on exit**.
7. Click **OK** to save the changes.
8. Click **Yes** to accept the certificate.
9. In Access Manager Auto-Configuration of SAML 2, specify the Access Manager details:

   Access Manger Administration Console

   Specify the Access Manger URL with the full DNS.

   For example,

   `https://<Access Manager DNS><port>`

   The default port is 8443.

   Access Manger Administrator Credentials

   Specify the username and password of the Access Manager administrator in LDAP format.

   For example,

   Username: `cn=admin,o=novell`

   Authentication Server Administrator Credintials

   Specify the username and password of the User Application administrator. For example, **Username**: `uaadmin`
10. Click **OK**.
11. Click **Yes** to accept the certificate.

12 Click **Yes** to continue.

Displays the Access Manager SAML 2 configuration summary.

13 Click **OK**.

14 Restart the Tomcat instance that hosts OSP.

Updating the Login Pages for Access Manager

The default login pages for Access Manager use HTML iFrame elements that conflict with the elements used for the identity applications. This section provides instructions for eliminating that conflict by creating a new login method and contract for Access Manager. The `.jsp` files referenced in this section are located by default in the `/opt/novell/nam/idp/webapps/nidp/jsp` directory on Linux. On Windows, they are located by default in the `C:\Program Files (x86)\Novell\Tomcat\webapps\nidp\jsp` directory.

For more information, see "Customizing the Identity Server Login Page" in the *NetIQ Access Manager Administration Guide*.

1 Modify the `top.jsp` file according to TID 7004020 and TID 7018468.

2 (Optional) For backup purposes, copy and rename the `login.jsp` file. For example, rename it to `idm_login.jsp`.

3 Open the Administration Console for Access Manager.

4 Create a new user store to connect to Identity Vault.

4a Click **Devices > Identity Servers > Edit > Local > User Stores**.

4b Click **New** and specify the required Identity Vault details:

Name

Specify the DNS of the Identity Vault.

Admin Name

Specify the Identity Vault administrator name in the LDAP format.

Admin Password

Specify the Identity Vault administrator password.

Directory type

Select eDirectory from the list.

Server Replica

1. Click **New** and specify **Name and IP Address/DNS Name** of Identity Vault.
2. Check **Use Secure LDAP connections**.
3. Click **Auto import trusted root** to import the Identity Vault certificate.
4. Click **OK**.

Search Contexts

1. Click **New**.
2. In **Search Context**, specify the search container.
3. In **Scope**, select **Subtree**.
4. Click **OK**.

5 To create a new login method, complete the following steps:

5a Click **Devices > Identity Servers > Edit > Local > Methods**.

5b Click **New**, then specify the **Display Name** for the new method. For example, `IDM Name/Password`.

5c For **Class**, specify **Name/Password-Form**.

5d For **User Store**, specify Identity Vault as an LDAP user store.

5e In the **Properties** section, click **New**, then specify the following properties:

Name	Value
JSP	idm_login
MainJSP	true

5f Click **OK**.

5g Click **Finish**.

6 To create a contract that uses the new login method, complete the following steps:

6a Click **Contracts > New**.

6b In the **Configuration** tab, specify the **Display Name** for the new contract. For example, `IDM Name/Password`.

6c For **URI**, specify `name/password/uri/idm`.

6d Under **Methods**, add the method that you created in Step 5. For example, `IDM Name/Password`.

6e Click **Next**.

6f In the **Authentication Card** tab, specify an **ID** for the card. For example, `IDM_NamePassword`.

6g Specify an image for the card.

6h Click **Finish**.

7 To specify the default values for how the system processes the new authentication contract, complete the following steps:

7a On the **Local** tab, click **Defaults**.

7b For User Store, specify Identity Vault as an LDAP user store.

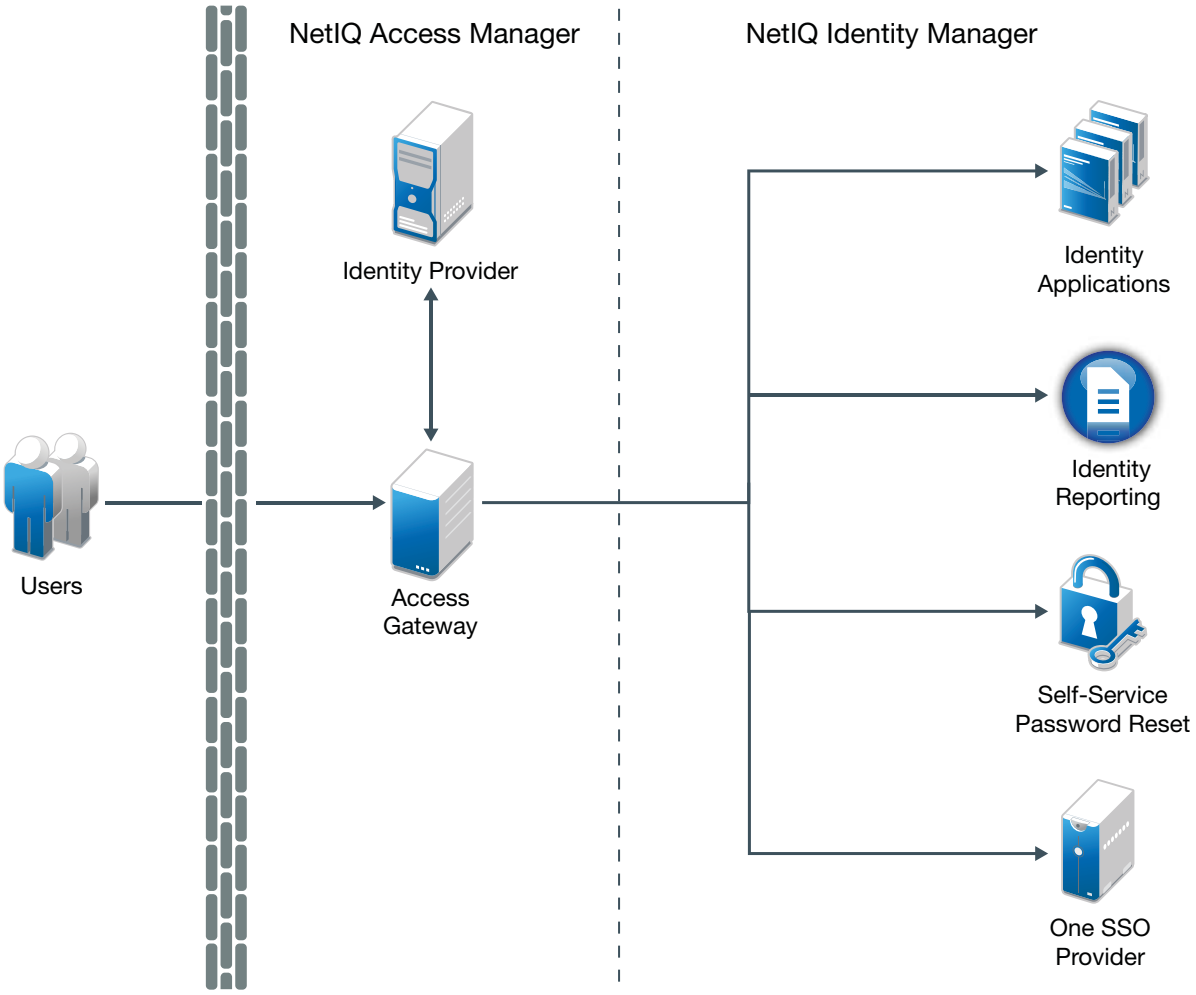
7c For **Authentication Contract**, specify the contract that you created in Step 6. For example, `IDM Name/Password-Form`.

7d Click **OK**.

8 To update the Identity Server, click **Devices > Identity Servers > Update > Update All Configuration**.

Reverse Proxy Based Single Sign-On

You can configure Access Manager to provide protected access to the Identity Manager web resources (identity applications and Identity Reporting) by using a domain-based proxy service and single sign-on access by using a form fill policy.



You can either use an existing reverse proxy and add a new proxy service for protecting the web resources or configure a new reverse proxy. While configuring the reverse proxy, create domain-based services for the servers hosting OSP and SSPR, identity applications, and Identity Reporting to enable single sign-on. You must configure these web resources as protected resources and specify the authentication procedures and the policies that should be used to enforce protection.

This section discusses a domain based proxy-service method with an example configuration. For more information about these proxy-service methods, see the *NetIQ Access Manager 4.4 Administration Guide*. Use information from the following table to understand the configuration required for different deployment scenarios.

Table 35-1 *Sample Configuration*

Deployment Scenario	Published DNS Name	Web Server Host Name	HTML Rewriting	Protected Resources	Form Fill
All components on the same server	rbpm.mycompany.com	rbpm.privatedns.com	Enable the following: ◆ Rewrite Inbound Query String Data ◆ Rewrite Inbound Headers ◆ Enable Rewrite Actions	/* Or /osp/* /sspr/* /idmdash/* /idmadmin/* /IDMPROV/*	Page Matching Criteria `<title>NetIQ Access</title>` Required only for OSP.
OSP and SSPR on a different server	osp.mycompany.com	osp.privatedns.com	Enable the following: ◆ Rewrite Inbound Query String Data ◆ Rewrite Inbound Headers ◆ Enable Rewrite Actions	/osp/* /sspr/*	Page Matching Criteria `<title>NetIQ Access</title>` Required only for OSP.
Identity Applications on a different server	identityapplications.mycompany.com	identityapplications.privatedns.com	Enable the following: ◆ Rewrite Inbound Query String Data ◆ Rewrite Inbound Headers ◆ Enable Rewrite Actions	/idmdash/* /idmadmin/* /IDMPROV/*	

Deployment Scenario	Published DNS Name	Web Server Host Name	HTML Rewriting	Protected Resources	Form Fill
Identity Reporting on a different server	RPT.mycompany.com	RPT.privatedns.com	Enable the following: ◆ Rewrite Inbound Query String Data ◆ Rewrite Inbound Headers ◆ Enable Rewrite Actions	/IDMRPT/* /IDMDCS/*	

You must be familiar with NetIQ Access Manager capabilities to understand the context of the content in this section. For more information about NetIQ Access Manager, see the Access Manager documentation website.

- Creating and Configuring the Proxy Service
- Creating Protected Resources
- Creating and Assigning a Form Fill Policy to a Protected Resource
- Configuring a Rewriter Profile
- Configuring Identity Providers
- Configuring Additional Redirect URLs in OSP Configuration File
- Testing the Single Sign-On

Creating and Configuring the Proxy Service

This section assumes that you have installed:

- NetIQ Access Manager and a NetIQ Access Manager Access Gateway
- All identity applications components on a single server (first scenario in Table 35-1)

You will first create a reverse proxy, for example `rbpm`, and then configure it to include domain-based multi-homed proxy services.

Remember that for the Web Server IP Address setting of the proxy service, you need to specify the IP Address for the identity applications server, and for the Web Server Host Name setting of the proxy service, you need to specify the DNS name of the identity applications web server.

1. Log in to the Administration Console. For example, `https://idmnam.acmeinfotech.com/nps`.
2. Click **Devices > Access Gateways > AG-Cluster > NAM-RP**.

3 Generate a certificate key by using the Access Manager CA:

3a Click **Auto-generate Key**, then click **New** twice.

3b Specify a name for the certificate. For example, `certificate_proxy_wildcard`.

3c Click **Edit** on the **Subject** line and specify **Common name** as `*.acmeinfotech.com`.

3d Click **OK** twice, then click the name of the certificate.

3e Click **OK** to ensure that all CA certificate chains of the selected certificate are added to the appropriate trust stores.

4 In the **Proxy Service List** section, click **New**.

4a Specify the following information:

Proxy Service Name: Specify a name that intuitively identifies this service on your Access Gateway server. For example, specify `rbpm`.

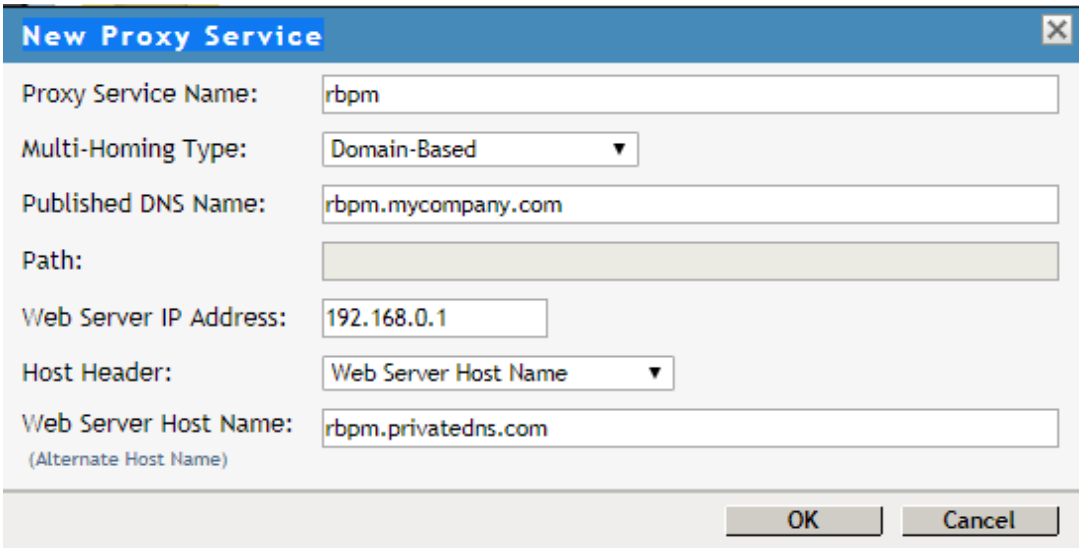
Multi-Homing Type: Select **Domain-Based**. The Access Gateway will use this method to identify the proxy service.

Published DNS Name: Specify the DNS name you want the public to use to access the web server. This DNS name must resolve to the IP address you set up as the listening address. For example, `rbpm.mycompany.com`

Web Server IP Address: Specify the IP address of the web server you want this proxy service to manage. You can specify additional Web server IP addresses by clicking the **Web Server Addresses** link when you have finished creating the proxy service.

Host Header: Specify the **Web Server Host Name** option. This allows the HTTP header to contain the published DNS name.

Web Server Host Name: Specify the DNS name of the server hosting the application to which the Access Gateway should forward requests. This entry matches what you specify in the Published DNS Name field.



NOTE: If OSP and Identity Applications are installed in a distributed environment, perform these actions for each service. In this case, the published DNS Name will be different for each service.

4b Click **OK**.

5 Click the newly added proxy service (**rbpm**), then select the **Web Servers** tab.

The Web servers added to this list must contain identical web content. Configuring your system with multiple servers with the same content adds fault tolerance and increases the speed for processing requests.

For caching to work correctly, the web servers must be configured to maintain a valid time. They should be configured to use an NTP server.

5a To enable SSL connections between the proxy service and its web servers, select **Connect Using SSL** and select **Do not verify** or **Import SSL Mutual Certificate** for the **Web Server Trusted Root** option.

Use **Do not verify** when you want the information between the Access Gateway and the identity applications server encrypted, but you do not need the added security of verifying the certificate of the identity applications server.

Use **Import SSL Mutual Certificate** to set up mutual authentication so that the identity applications server can verify the proxy service certificate.

5b In the **Connect Port** field, specify the port that the web server uses for SSL communication. This is the port that the identity applications server is listening from Access Gateway. For example, by default, the listening port is 8180.

If the identity applications server is listening on an SSL port, ensure that you specify that port and enable **Connect Using SSL**.

If identity applications are listening on a non-SSL port, ensure that you configure that port and verify that **Connect Using SSL** is disabled.

5c Leave the other settings unchanged.

5d Click **OK**.

5e Under **Access Gateway Servers**, click **Update All** for AG-Cluster to apply changes of reverse proxy service created.

Creating Protected Resources

After creating the proxy service, create protected resources with resource paths to all applications that are configured. The protected resource configuration specifies the authentication procedures and the policies that should be used to enforce protection.

To create a resource for the identity applications:

1 Log in to the Access Manager Administration Console.

2 In the console, click **Devices > Access Gateways > Edit > [Name of Reverse Proxy] > [Name of Domain-Based Proxy Service or Primary Proxy Service] > Protected Resources**.

3 Create a protected resource.

3a Select **New**, then specify a display name for the new resource you want to protect.

For example, to create a resource that you want to use to represent all identity applications resources, you can name the resource as `osp_protected_resources`.

When you create the display name, the Overview page for the new resource is displayed.

3b Fill in the fields to configure the resource:

Description: Specify a description for the protected resource. You can use it to briefly describe the purpose for protecting this resource.

Authentication Procedure: Select **Name/Password -Form** from the drop-down list. This specifies a form-based authentication over HTTP or HTTPS, using the Access Manager login form.

URL Path List: Remove the `/*` path and add paths for Published DNS Name, `rbpm.mycompany.com`.

For example, include the following paths for OSP and SSPR protected resource:

```
/osp/*
```

```
/sspr/*
```

For the identity applications protected resource, include the following paths:

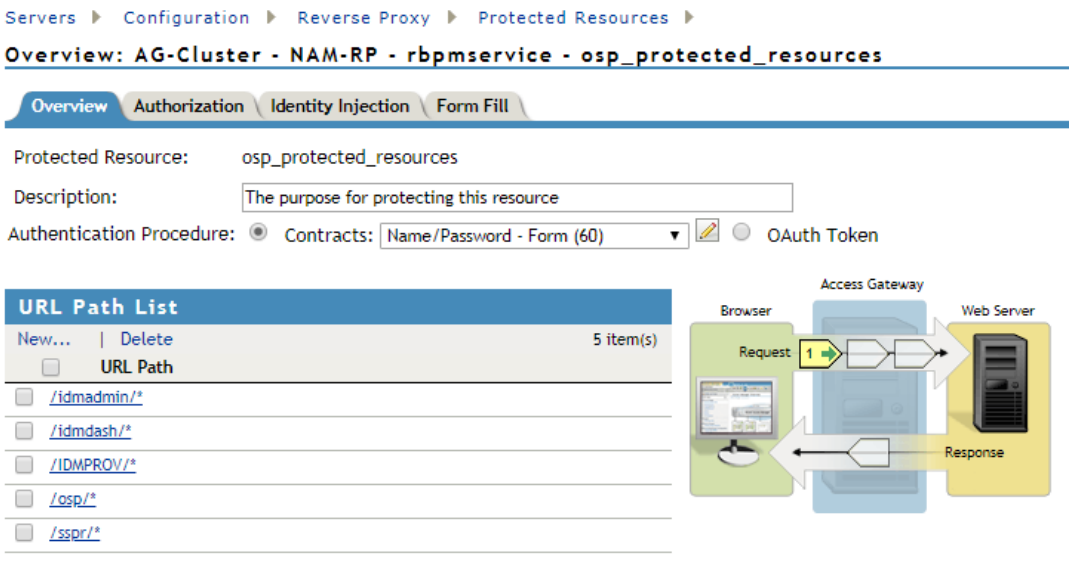
```
/idmdash/*
```

```
/idmadmin/*
```

```
/IDMPROV/*
```

For Identity Reporting protected resource, include the following path:

```
/IDMRPT/*
/IDMDCS/*
```



Click **OK**.

3c Click the Protected Resources breadcrumb at the top of the Overview page to return to the Protected Resources page.

3d In the **Protected Resource List**, ensure that the protected resources you created are enabled.

3e To apply your changes, click **Devices > Access Gateways**, then click **Update**.

4 Continue with Creating a Form Fill Policy for the newly created protected resources.

Creating and Assigning a Form Fill Policy to a Protected Resource

You must create a Form Fill policy and assign it to the OSP protected resource. This Form Fill policy will match the OSP login form and fill the user credentials for single sign-on. The policy also includes a failure policy which will redirect the identity applications logout request to also do the Access Gateway logout.

Form Fill policy must be applied to only the OSP protected resource and not for identity applications or Identity Reporting protected resources because these applications will be automatically redirected to the OSP login page if a user is not logged in.

To create a Form Fill Policy for the OSP protected resource:

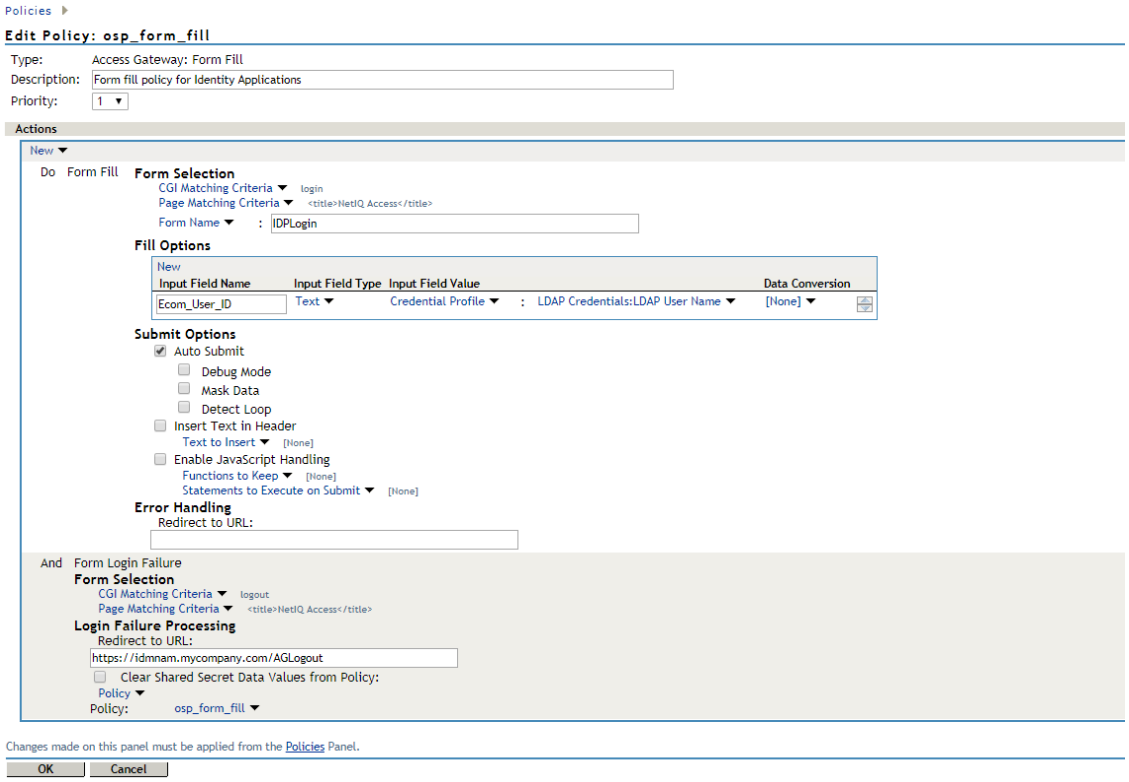
1 In the Administration Console, click **Policies > Policies**.

2 Select the policy container, then click **New**.

3 Specify a name for the policy, select **Access Gateway: Form Fill** as the type of the policy, then click **OK**. For example, `osp_form_fill`

4 (Optional) Specify a description for the Form Fill policy. This is useful if you plan to create multiple Form Fill policies.

5 In the **Actions** section, click **New**, then select **Form Fill**.



6 In the **Form Selection** section, perform the following actions:

6a Select **Form Name** Specify the name of the form. For example, `IDPLogin`.

6b Select **Page Matching Criteria**. This causes the Access Gateway to search the HTML page for the specified text. If the specified text is found on the page, the page is a match for the policy. If it is not found, the page is not a match for the policy and the policy is not applied. For example, suppose your HTML page has the following string within the <FORM> element:

```
<title>NetIQ Access</title>
```

If you enter this string in the Page Matching Criteria box, the Access Gateway searches the form for this string. If it finds the string, it knows it has a match (get this unique tag from page source of OSP Login screen).

7 In the **Fill Options** section, create an entry for all the input fields and select options in the form. For each input field or select option, you need to specify the following information:

- **Input Field Name:** Specifies the name of the field or option. This is the name attribute of the element on the form. For example, `Ecom_User_ID`
- **Input Field Type:** Specifies the type attribute for the input field or select option in the form. Select **Text**.
- **Input Field Value:** Specify the value for the field. You must specify the data type, then enter the value. Select **Credential Profile: LDAP Credentials: LDAP User Name**

7a Click New and specify the values for the following fields:

- **Input Field Name:** `Ecom_Password`
- **Input Field Type:** **Password**

- **Input Field Value:** Credential Profile: LDAP Credentials: LDAP User Name

8 In the Submit Options section, select Auto Submit.

This action indicates you want the form submitted to the Web server without having the user confirm the submission by clicking a Submit button. If this option is not selected, Form Fill can fill in the data, but the user must click the Submit button before the data is sent to the Web server. When the form is not auto submitted, all the JavaScript on the form is executed.

9 To create a login failure policy, click New in the Actions section, then select Form Login Failure.

10 In the Form Selection section, perform the following actions:

10a Select CGI Matching Criteria. This allows the Access Gateway to evaluate the query string in the URL (the portion after the question mark) to differentiate pages that have the same URL. Type `logout`.

11 In the Login Failure Processing section, specify the Access Gateway URL in the following field:

- **Redirect to URL:** Specify `https://idmnam.mycompany.com/AGLogout`.

  When an LDAP or NSS error occurs, the user is redirected to the URL you specify in the text box.

12 Use the up-arrow button to move the Form Login Failure policy to the top of the policy list.

You want the failure policy to execute first on login failure.

13 Click OK.

14 Click OK, then click Apply Changes and close the Policies window.

15 To enable the policy you just created (`osp_form_fill`), click Enable.

16 Click OK in the Protected Resources tab.

17 Continue with configuring HTML rewriting for the proxy service.

Configuring a Rewriter Profile

The changes you make to the NetIQ Access Gateway configurations for the protected resources require HTML rewriting because the web servers hosting the protected resources are not aware that the Access Gateway system is obfuscating their DNS names. URLs contained in these pages must be checked to ensure that these references contain the DNS names that the client browser understands. On the other end, the client browsers are not aware that the Access Gateway is obfuscating the DNS names of the resources they are accessing. The URL requests coming from the client browsers that use published DNS names must be rewritten to the DNS names that the identity applications web servers expect. For more information, see Configuring HTML Rewriting in the *NetIQ Access Manager 4.4 Administration Guide*.

When you configure the HTML rewriter for a proxy service, these values are applied to all the resources protected by this proxy service. Keeping everything as default, enable Rewrite Inbound Query String Data, Rewrite Inbound Headers, and Enable Rewrite Actions in this profile and then move this custom profile above the Default profile and save the configuration. Perform this action for all the services in a distributed setup.

1 Log in to the Access Manager Administration Console.

2 Click Devices > Access Gateways > Edit > [Name of Reverse Proxy] > [Name of Proxy Service] > HTML Rewriting.

3 Click HTML Rewriter Profile List to create a profile.

New: Specify a display name for the profile. For example, `idmnamrewriter_custom`.

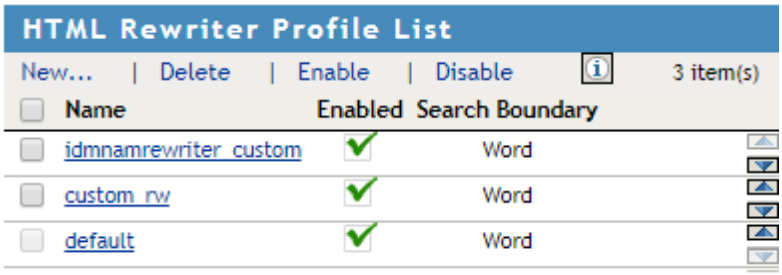
Word: Select **Word** for **Search Boundary**. A **Word** profile searches for matches on words. For example, “get” matches the word “get” and any word that begins with “get” such as “getaway” but it does not match the “get” in “together” or “beget.”

4 Navigate to the rewriter profile that you created from **HTML Rewriter** and ensure the following fields are selected:

- **Rewrite Inbound Query String Data**
- **Rewrite Inbound Headers**
- **Enable Rewrite Actions**

5 Click **OK**.

6 In **HTML Rewriter Profile List**, move the Word profile to be the first profile in the list.



7 To save your changes, click **OK**.

8 To update the Access Gateway, the cached pages affected by the rewriter changes must be updated on the Access Gateway. Click **OK** to purge all cache.To apply your changes, click the Access Gateways link, then click **Update > OK**.

Configuring Identity Providers

User stores are LDAP directory servers to which end users authenticate. You must specify an initial user store when creating an Identity Server configuration. You use the same procedure for setting up the initial user store, adding a user store, or modifying an existing user store.

1 Log in to NetIQ Access Management Administration Console. For example, `https://idmnam.mycompany.com/nps`.

2 Navigate to **Devices > Identity Servers > AG-Cluster > Edit > Local**.

3 In **User Stores**, click **New** to add a user store. This is the Identity Vault used by your Identity Manager.

4 Fill in the following fields:

Name: The name of the user store for reference. For example, `acmeidm`

Admin Name: The distinguished name of the admin user of the LDAP directory, or a proxy user with specific LDAP rights to perform searches. For the LDAP extension to work, the proxy user requires write rights on the ACL. Administrator-level rights are required for setting up a user store. This ensures read/write access to all objects used by Access Manager.

For example, `cn=admin,ou=sa,o=system`

Admin Password and Confirm Password: Specify the password for the admin user and confirm it.

Directory Type: The type of LDAP directory. Select eDirectory.

If eDirectory has been configured to use Domain Services for Windows, eDirectory behaves like Active Directory. When you configure such a directory to be a user store, its Directory Type must be set to Active Directory for proper operation.

5 To specify a server replica, click **New**, then fill in the following fields:

For an eDirectory server, you must use a replica of the partition where the users reside. Ensure that each LDAP server in the cluster has a valid read/write replica. One option is to create a users partition (a partition that points to the OU containing the user accounts) and reference this server replica.

Name: Specify the name of the Identity Vault used by the Identity Manager server.

IP Address: The IP Address of the Identity Vault used by the Identity Manager server. If your eDirectory is replicated on multiple servers, use this name to identify a specific replica.

Use secure LDAP connections: Specifies that the eDirectory server requires secure (SSL) connections with the Identity Server.

This is the only configuration we recommend for the connection between the Identity Server and the eDirectory server in a production environment. If you use port 389, usernames and passwords are sent in clear text on the wire.

Name: The display name for the eDirectory server. If your eDirectory is replicated on multiple servers, use this name to identify a specific replica.

6 To import the trusted root certificate from the eDirectory server, click **Auto import trusted root**.

7 Click **OK** to confirm the import.

The Root CA Certificate and Server Certificate are imported into the Access Manager server keystore.

8 To add a search context, click **New** under **Search Contexts** and define the search context **o=context** (for example: `o=data`) and define the search scope as `Subtree`.

The search context is used to locate users in the directory when a contract is executed.

9 Click **Finish**.

10 Navigate to **Local > Methods** and select the following authentication class:

Secure Name/Password - Form: Form-based authentication over HTTPS.

11 Locate the user store that you just created (`acmeidm`) under **User stores** and move it under **Available user stores**.

12 Click **Apply** and **OK**.

13 If prompted to restart Tomcat, click **OK**. Otherwise, update the Identity Server. Open the Identity Servers window **> Servers > Update All > IDPCluster** and apply the changes and ensure it is current and green.

Configuring Additional Redirect URLs in OSP Configuration File

1 Install the latest OSP version.

2 Stop the Tomcat services (`systemctl stop netiq-tomcat.service`).

3 Edit the `ism-configuration.properties` file from `tomcat/conf` and append '`additional.redirect.urls`' entries in the file based on the published DNS names. For this example, the entries can look like below:

```
ospservice:/opt/netiq/idm/apps/tomcat/conf # cat ism-
configuration.properties | grep redirect

com.netiq.idmdash.redirect.url = https://rbpm.mycompany.com:8543/
idmdash/oauth.html

com.netiq.idmdash.additional.redirect.urls = https://
rbpm.privatedns.com/idmdash/oauth.html

com.netiq.rbpm.redirect.url = https://rbpm.mycompany.com:8543/IDMProv/
oauth

com.netiq.rbpm.additional.redirect.urls = https://rbpm.privatedns.com/
IDMProv/oauth

com.netiq.rpt.redirect.url = https://RPT.mycompany.com:8543/IDMRPT/
oauth.html

com.netiq.rpt.additional.redirect.urls = https://RPT.privatedns.com/
IDMRPT/oauth.html

com.netiq.sspr.redirect.url = https://osp.mycompany.com:8543/sspr/
public/oauth

com.netiq.sspr.additional.redirect.urls = https://osp.privatedns.com/
sspr/public/oauth
```

4 Start the Tomcat service (`systemctl start netiq-tomcat.service`).

Testing the Single Sign-On

1 Open a browser and launch the NAM URL protecting the identity applications.

For example: `https://rbpm.privatedns.com/IDMProv/`

Ensure that the URLs are resolvable either through the host entries or DNS.

2 On the authentication prompt, provide the user credentials in Secure Name/Password - Form. For example, `uaadmin` and `password`.

3 Upon submitting the credentials you should be able to view Identity Applications Dashboard.

4 Access other pages also and you can observe the SSO experience.

For example: `https://rbpm.privatedns.com/idmadmin`

36 Using Kerberos for Single Sign-On

You can use Kerberos as an authentication method for the identity applications that allows single sign-on (SSO). This also allows users to use Integrated Windows Authentication to log in to the applications. This section provides instructions for configuring Active Directory to use Kerberos for connecting to the identity applications:

- "Configuring the Kerberos User Account in Active Directory" on page 563
- "Configuring the Identity Applications Server" on page 564
- "Configure the End-User Browsers to Use Integrated Windows Authentication" on page 566
- "Logging In Using the Name Password Form" on page 567

Configuring the Kerberos User Account in Active Directory

Use the Active Directory administration tools to configure Active DIrectory for Kerberos authentication. You need to create a new Active Directory user account for the identity applications and identity reporting. The user account name must use the DNS name of the server that hosts the identity applications and identity reporting.

NOTE: For domain or realm references, use uppercase format. For example `@MYCOMPANY.COM`.

---

1 As an Administrator in Active Directory, use the Microsoft Management Console (MMC) to create a new user account with the DNS name of the server that hosts the identity applications.

   For example, if the DNS name of the identity applications server is `rbpm.mycompany.com`, use the following information to create the user:

   First name: rbpm

   User login name: HTTP/rbpm.mycompany.com

   Pre-windows logon name: rbpm

   Set password: Specify the appropriate password. For example: `Passw0rd`.

   Password never expires: Select this option.

   User must change password at next logon: Do not select this option.

2 Associate the new user with the Service Principal Name (SPN).

   2a In the Active Directory server, open a cmd shell.

   2b At the command prompt, enter the following:

   ```
   setspn -A HTTP/DNS_Identity_Applications_server@WINDOWS-DOMAIN
   userID
   ```

   For example:

   ```
   setspn -A HTTP/rbpm.mycompany.com@MYCOMPANY.COM rbpm
   ```

   2c Verify setspn by entering `setspn -L userID`.

3 To generate the keytab file, use the ktpass utility:

3a At the command line prompt, enter the following:

```
ktpass /out filename.keytab /princ servicePrincipalName /mapuser
userPrincipalName /mapop set /pass password /crypto ALL /ptype
KRB5_NT_PRINCIPAL
```

For example:

```
ktpass /out rbpm.keytab /princ HTTP/rbpm.mycompany.com@MYCOMPANY.COM /
mapuser rbpm  /mapop set /pass Passw0rd /crypto All /ptype
KRB5_NT_PRINCIPAL
```

IMPORTANT: For domain or realm references, use uppercase format. For example, @MYCOMPANY.COM.

3b Copy the rbpm.keytab file to your identity applications server.

4 As an Administrator in Active Directory, create an end user account with the MMC to prepare for SSO.

The end user account name has to match some attribute value of an eDirectory user in order to support single sign-on. Create the user with some name such as cnano, remember the password, and ensure that **User must change password at next logon** is not selected.

5 (Optional) Repeat these steps for Identity Reporting if you installed the reporting component on a separate server.

6 Configure the server for the identity applications to accept the Kerberos configuration. For more information, see "Configuring the Identity Applications Server" on page 564.

Configuring the Identity Applications Server

You must configure your identity applications server to use the Kerberos keytab file and the user account that you have created in Active Directory. Ensure that you complete "Configuring the Kerberos User Account in Active Directory" on page 563 before proceeding.

NOTE: For domain or realm references, use uppercase format. For example @MYCOMPANY.COM.

---

1 To define your operating system settings for the Kerberos configuration, complete the following steps:

1a Open the krb5 file in a text editor on the server that hosts the identity applications.

Linux: /etc/krb5.conf

Windows: C:\Windows\krb5.ini

UNIX: /etc/krb5/krb5.conf

1b Add the following information to the krb5 file:

```
[libdefaults]
    default_realm = WINDOWS-DOMAIN
    kdc_timesync = 0
    forwardable = true
    proxiable = false
[realms]
    WINDOWS-DOMAIN = {
         kdc = FQDN Active Directory Server
         admin_server = FQDN Active  Directory Server
    }
[domain_realm]
    .your.domain = WINDOWS-DOMAIN
    your.domain = WINDOWS-DOMAIN
```

For example:

```
[libdefaults]
    default_realm = MYCOMPANY.COM
    kdc_timesync = 0
    forwardable = true
    proxiable = false
[realms]
    MYCOMPANY.COM = {
         kdc = myadserver.mycompany.com
         admin_server = myadserver.mycompany.com
    }
[domain_realm]
    .mycompany.com = MYCOMPANY.COM
    mycompany.com = MYCOMPANY.COM
```

1c Save the changes and close the `krb5` file.

2 (Conditional) To define the Kerberos configuration information for Tomcat, complete the following steps:

2a Create a sample `Kerberos_login.config` file on the Tomcat application server with the following content:

NOTE: The novlua user needs permissions to create the `Kerberos_login.config` file.

```
com.sun.security.jgss.krb5.accept {
        com.sun.security.auth.module.Krb5LoginModule required
    debug="true"
        refreshKrb5Config="true"
    useTicketCache="true"
        ticketCache="/opt/netiq/idm/apps/tomcat/kerberos/
spnegoTicket.cache"
    doNotPrompt="true"
        principal="HTTP/DNS_Identity_Applications_server@WINDOWS-
DOMAIN"
    useKeyTab="true"
        keyTab="/absolute_path/filename.keytab"
    storeKey="true";
    };
```

An example on a Windows server is as follows:

```
keyTab="c:\\NetIQ\\IdentityManager\\apps\\tomcat\\kerberos\\rbpm.keytab"
```

2b In the file, specify values for `principal` and `keyTab`. For example:

```
principal="HTTP/rbpm.mycompany.com@MYCOMPANY.COM"
keyTab="/home/usr/rbpm.keytab"
```

- The value for `principal` must match the same value that you specified for Kerberos. For more information, see Step 3 on page 564.
- Provide the absolute path of the `keytab` file on your identity applications server. The file does not have to reside in the default directory for the identity applications.

2c Refer to the `Kerberos_login.config` file in JVM `java.security` file with the following line:

```
login.config.url.1=file:/opt/netiq/idm/apps/tomcat/kerberos/
Kerberos_login.config
```

The path listed is the default installation location for a Linux server.

An example of the `java.security` file on a Windows server is as follows:

```
login.config.url.1=file:c:\NetIQ\IdentityManager\apps\tomcat\kerberos\Kerberos_login.config
```

3 To specify the Authentication method in the RBPM Configuration utility, complete the following steps:

3a Open the `Configupdate` utility.

3b Click the **Authentication** tab.

3c Scroll down to the **Authentication Method** section.

3d In the **Method** field, select **Kerberos**.

3e In the **Mapping attribute name** field, specify `cn`.

4 (Optional) Repeat these steps for Identity Reporting if you installed the reporting component on a separate server.

5 Configure the browsers that end-users use to access the identity applications. For more information, see "Configure the End-User Browsers to Use Integrated Windows Authentication" on page 566.

Configure the End-User Browsers to Use Integrated Windows Authentication

The browsers that your end-users use to access the identity applications and identity reporting also need to be configured for Integrated Windows Authentication. This section provides instructions for configuring an end-user computer to support single sign-on access using Integrated Windows Authentication.

NOTE: You must perform this procedure for each end-user computer where you want to provide single sign-on access to the identity applications and identity reporting.

1. Log in to the computer where users will need single sign-on access.
2. Open the Internet options control panel.
3. Click **Security**.
4. Click **Trusted Sites** > **Sites**.
5. Add the DNS name of the identity applications server.

   For example: `rbpm.mycompany.com`
6. Click **Add**, then click **Close**.
7. Click **Custom level...**.
8. Under **User Authentication**, select **Automatic logon with current user name and password**.
9. Click **OK**.
10. In Internet Options, click **Advanced**.
11. Under Security, select **Enable Integrated Windows Authentication**.
12. Repeat this procedure for each end-user computer where you want to provide single sign-on access to the identity applications and identity reporting.

Logging In Using the Name Password Form

When Identity Applications are configured for Kerberos authentication, the login process uses Kerberos contract by default. If you want to log in using the `name-password` contract instead of the default contact, use the following URL:

```
https://<ospserver:port>/osp/a/idm/auth/app?acAuthCardId=np-contract-
$default-card$&target=https://<rbpm server:port>/idmdash/
```

The logout process relogins you to the system as Kerberos logged-in user. You need to logout to come out of the identity applications.

37 Integrating Single Sign-on Access with Identity Governance

If you have installed Identity Manager, your users can log in a single time to access Identity Applications, Identity Reporting, and Identity Governance from the Identity Manager Home page. To ensure single sign-on access, you must configure both Identity Manager and Identity Governance for single sign on. Users can easily shift between the two applications without needing to enter their credentials a second time. Identity Governance must use the same authentication server that the identity applications use.

This chapter includes the following topics:

- "Ensuring Rapid Response to Authentication Requests" on page 569
- "Configuring Identity Governance for Integration" on page 570
- "Configuring Identity Manager for Integration" on page 573

Ensuring Rapid Response to Authentication Requests

You can configure OSP so users can log in with an email address or another attribute available in the Identity Vault. If you use a non-default attribute, the server might take longer to respond to authentication requests. Also, OSP automatically times out LDAP connections after 15 seconds. To ensure a rapid response time, the LDAP authentication server should have an index for the login attribute. You also must specify that attribute in the RBPM Configuration Utility.

1 To specify the login attribute, complete the following steps:

   1a Run the RBPM Configuration utility.

   For more information, see Configuring the Settings for the Identity Applications in the *NetIQ Identity Manager Setup Guide for Linux* or Configuring the Settings for the Identity Applications in the *NetIQ Identity Manager Setup Guide for Windows*.

   1b Select **Authentication > Show Advanced Options**.

   1c For **Duplicate resolution naming attribute**, specify the attribute that you want to use for login activities. For example, `Internet Email Address`.

   1d Save your changes.

2 (Conditional) To create an index for the login attribute in the Identity Vault, complete the following steps:

   2a Create the index.

   For more information, see Creating Compound Indexes in *NetIQ Identity Manager Setup Guide for Windows*.

   2b For the attribute, select the same attribute that you specified for **Duplicate resolution naming attribute** in the configuration utility.

2c For the index rule, specify **Value**.

2d Complete the process for creating the index.

Configuring Identity Governance for Integration

For proper integration, you must link Identity Governance to the Identity Manager Home page for the identity applications. You can also choose to use the same authentication server that the identity applications use to verify login attempts. This process includes the following activities:

- "Adding a Link for Identity Manager Home in the Identity Governance Menu" on page 570
- "Using the Same Authentication Server as Identity Manager" on page 570
- "Registering Identity Applications Server" on page 572

Adding a Link for Identity Manager Home in the Identity Governance Menu

This section describes how to add a link in Identity Governance so users can easily switch to Identity Manager Home.

1 Log in to Identity Governance with an account that has the Global Administrator authorization.

2 Select **Configration > General Settings**.

3 For **Home Page URL**, specify the URL for Identity Manager Home. For example, `https://myserver:8543/idmdash#/landing`.

4 Select **Save**.

5 Refresh the browser for change to take effect.

Using the Same Authentication Server as Identity Manager

This section describes how to configure Identity Governance to use the same authentication server as Identity Manager identity applications for verifying users who log in. This section assumes that, when you installed Identity Governance, you did not specify the Identity Manager authentication server. For example, you might have installed Identity Governance before adding Identity Manager to your environment.

NOTE: Identity Applications use `https` communication by default. You should create a *wildcard certificate* on one of the servers and copy the certificate on all the servers.

For example, the wildcard certificate `*.example.com` is created on OSP server.

1 Add this certificate to the `keystoreFile` on all the servers.

2 Restart Tomcat on all the servers.

Ensure that `keystoreFile` is updated in the `server.xml`.

```
<Connector port="8543"
protocol="org.apache.coyote.http11.Http11NioProtocol" maxThreads="150"
SSLEnabled="true" scheme="https" secure="true" clientAuth="false"
sslProtocol="TLSv1.2" keystoreFile="conf/tomcat.ks" keystorePass="novell"
sslEnabledProtocols="TLSv1.2" />
```

1. Stop Tomcat.

   ```
   systemctl stop identity_tomcat.service
   ```

2. Navigate to the `/opt/netiq/idm/apps/idgov/bin` directory and run the `./configutil.sh -password <passwd>` command.
3. In the configuration utility (`configutil`), navigate to the **Authentication Server Details** tab and perform the following steps:
   1. De-select the **Same as IG Server** check box.
   2. Specify the authentication server details that is deployed with Identity Applications in the **Protocol**, **Host Name**, and **Port** fields.
   3. Set the value for the **Authentication Source** parameter from `File` to `Identity Vault` and then specify the LDAP user DN in the **Name** field. Specify the complete DN of the user. For example, `cn=uaadmin,ou=sa,o=data`.

      (Conditional) If you are using Identity Governance 3.6, then you must use the LDAP bootstrap user.

      (Conditional) If you are using Identity Governance 3.5.x, then setting the LDAP bootstrap user is optional based on the authentication method you used.
   4. Specify the values for the **Client ID** and **Redirect URL** under the **OAuth SSO Client** parameter.

      NOTE: The values for these settings must match the values that you specify for Identity Governance in the Identity Applications Configuration update utility.
   5. Click **Save** and close the configuration utility.
4. In the configuration utility (`configutil`), navigate to the **Security Settings** tab and make a note of the Client ID for all the clients.

   NOTE: The values for these settings must match the values that you specify for Identity Governance in the Identity Applications Configuration update utility.
5. Navigate to `/opt/netiq/idm/apps/configupdate` directory and run the `./configupdate.sh` command.
6. In the configuration update utility (`configupdate`), navigate to the **Authentication** tab and perform the following steps:
   1. Specify the authentication server details that is deployed with Identity Applications.
   2. Click **Show Advanced**.
   3. (Conditional) If you are using Identity Governance 3.6, you must select the **LDAP user** check box under the **Identity Governance Bootstrap Administrator** settings.

4. (Conditional) If you are using Identity Governance 3.5.x and the authentication method is set to Kerberos or SAML 2.0, then you must select the **LDAP user** check box under the **Identity Governance Bootstrap Administrator** settings. If the authentication method is set to **Name and Password**, you can select the file-based user or LDAP user as your bootstrap administrator.
5. Click **OK**.
6. (Conditional) If the Tomcat certificate of the server where Identity Applications is deployed has not been installed into the trust store, then you must accept the certificate. Ensure that Tomcat is running on the Identity Applications server.

7 Start Tomcat.

```
systemctl start identity_tomcat.service
```

Registering Identity Applications Server

You should register Identity Applications server details on Identity Governance server that allows Identity Applications to access Identity Governance through Identity Manager Dashboard.

Perform the following steps to register Identity Applications server:

1 Log in to Identity Governance server as an administrator.

2 Stop Tomcat.

```
systemctl stop identity_tomcat.service
```

3 Launch the configuration update utility in the console mode.

For example,

```
/opt/netiq/id/apps/idgov/bin/configutil.sh -password $db_password -
console
```

4 Run the following command to register the CORS client.

```
ap GLOBAL com.netiq.iac.CORSclient $URL_OF_RBPM_MACHINE
```

For example, `ap GLOBAL com.netiq.iac.CORSclient https://myserver.mydomain.com:8543`

5 Verify the value using the following command.

```
dc com.netiq.iac.CORSclient
```

After registering the Identity Applications server on the Identity Governance server, you should update the Identity Governance URL in the Identity Manager Dashboard to see the Identity Governance tasks.

1 Log in to Identity Manager Dashboard as an administrator.

2 Select **YourID** > **Settings** > **Customization**.

3 Select **General** from **Navigation items** and specify the **Identity Governance URL**.

For more information, see "Changing Identity Applications Client Settings" on page 131 and "Customizing the Views" on page 131.

Configuring Identity Manager for Integration

To ensure proper integration, you must install the appropriate version of Identity Applications to recognize Identity Governance. This procedure assumes that you have configured single sign-on for the identity applications.

1. On the server where you installed Identity Applications, log in as an administrator.
2. Stop Tomcat.
3. Based on the platform, download and extract the `Identity_Manager_4.7.4_Linux.zip` or `Identity_Manager_4.7.4_Windows.zip` file from the download site.
4. Navigate to the following directory based on your platform:
   - **Linux:** `<patch extracted location>/Identity_Manager_4.7.4_Linux/osp/ig`
   - **Windows:** `<extracted patch location>\Identity_Manager_4.7.4_Windows\osp\ig`
5. (Conditional) If you are using Identity Governance 3.6, copy the `uaconfig-ig36-defs.xml` file and place it in the `/opt/netiq/idm/apps/tomcat/conf/` directory.
6. (Conditional) If you are using Identity Governance 3.5, copy the `uaconfig-ig-defs.xml` file and place it in the `/opt/netiq/idm/apps/tomcat/conf/` directory.
7. From the `/opt/netiq/idm/apps/tomcat/conf/` directory, edit the `ism-configuration.properties` file and add the following lines:
   - `com.netiq.iac.bootstrapadmin.authsrc = bisadus`
   - `com.netiq.idm.osp.fileauthsrc.enabled = false`

   If you want to use the new jwt token, then add:

   `com.netiq.idm.osp.oauth.access-token-format.format = jwt`
8. Navigate to the `/opt/netiq/idm/apps/tomcat/bin` directory.
9. Edit the `setenv.sh` file and add the following line before `-Duser.language` entry in the file:

   `-Dcom.netiq.uaconfig.impl.custom.clients=path_to_conf_dir/uaconfig-ig36-defs.xml`

   For example, if you are using Identity Governance 3.6, add the following line:

   `-Dcom.netiq.uaconfig.impl.custom.clients=/opt/netiq/idm/apps/tomcat/conf/uaconfig-ig36-defs.xml`
10. Save and close the `setenv.sh` file.
11. Launch the configuration update utility by running `./configupdate.sh` from the command prompt.
12. Navigate to the **Authentication** tab and click **Show Advanced Options**.
13. (Conditional) If you are using Identity Governance 3.6, you must select the **LDAP user** check box under the **Identity Governance Bootstrap Administrator** settings.
14. (Conditional) If you are using Identity Governance 3.5.x and the authentication method is set to Kerberoes or SAML 2.0, then you must select the **LDAP user** check box under the **Identity Governance Bootstrap Administrator** settings. If the authentication method is set to **Name and Password**, you can select the file-based user system or the LDAP user as your bootstrap administrator.

15 Navigate to the **IG SSO Clients** tab.

16 (Conditional) If you want to change the authentication server for Identity Governance after installation, specify the values based on the settings that you specified in Step 3 and Step 4 of the "Using the Same Authentication Server as Identity Manager" on page 570 section.

The following considerations apply to these settings:

- (Conditional) If you are using Identity Governance 3.6, the default **OAuth client ID** is `ig`.

NOTE: The client secret is specified during the Identity Governance installation. You can change the client ID and client secret after installation.

Specify the following details for **Identity Governance Client**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for Identity Governance to the authentication server. The default value is `ig`.
- **OAuth Redirect URI:** The **OAuth redirect URL** must be an absolute URL and include the specified value for OAuth client ID. For example, `https://myserver.host:8443/oauth.html`. By default, the configuration update utility provides some of this URL. However, you must ensure that you add the server and port information.

Specify the following details for **Identity Governance Utility Client**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for Identity Governance Utility to the authentication server. The default value is `iac`.
- **OAuth client secret:** Specifies the password for the single sign-on client for the Identity Governance utility client.

Specify the following details for **Request Client**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for Request to the authentication server. The default value is `cx`.
- **OAuth Redirect URI:** The **OAuth redirect URL** must be an absolute URL and include the specified value for OAuth client ID. For example, `https://myserver.host:8443/cx/oauth.html`. By default, the configuration update utility provides some of this URL. However, you must ensure that you add the server and port information.

Specify the following details for **Data Connectivity Service**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for Data Connectivity Service to the authentication server. The default value is `iac-daas`.
- **OAuth client secret:** Specifies the password for the single sign-on client for the Data Connectivity Service client.

Specify the following details for **General Service**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for General Service to the authentication server. The default value is `iac-service`.
- **OAuth client secret:** Specifies the password for the single sign-on client for the General Service client.

Specify the following details for **Data Transformation and Processing Service**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for Data Transformation and Processing Service to the authentication server. The default value is `iac-dtp`.
- **OAuth client secret:** Specifies the password for the single sign-on client for the Data Transformation and Processing Service client.

Specify the following details for **Workflow Service**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for Workflow Service to the authentication server. The default value is `wf`.
- **OAuth client secret:** Specifies the password for the single sign-on client for the Workflow Service client.

- (Conditional) If you are using Identity Governance 3.5.x, the default **OAuth client ID** is `iac`.

NOTE: The client secret is specified during the Identity Governance installation. You can change the client ID and client secret after installation.

Specify the following details for **Identity Governance Client**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for Identity Governance to the authentication server. The default value is `iac`.
- **OAuth client secret:** Specifies the password for the single sign-on client for the Identity Governance client.
- **OAuth Redirect URI:** The **OAuth redirect URL** must be an absolute URL and include the specified value for OAuth client ID. For example, `https://myserver.host:8443/oauth.html`. By default, the configuration update utility provides some of this URL. However, you must ensure that you add the server and port information.

Specify the following details for **Request Client**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for Request to the authentication server. The default value is `cx`.
- **OAuth client secret:** Specifies the password for the single sign-on client for the Request client.
- **OAuth Redirect URI:** The **OAuth redirect URL** must be an absolute URL and include the specified value for OAuth client ID. For example, `https://myserver.host:8443/cx/oauth.html`. By default, the configuration update utility provides some of this URL. However, you must ensure that you add the server and port information.

Specify the following details for **Data Connectivity Service**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for Data Connectivity service to the authentication server. The default value is `iac-daas`.
- **OAuth client secret:** Specifies the password for the single sign-on client for the Data Connectivity Service client.

Specify the following details for **General Service**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for General Service to the authentication server. The default value is `iac-service`.

- **OAuth client secret:** Specifies the password for the single sign-on client for the General Service client.

Specify the following details for **Data Transformation and Processing Service**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for Data Transformation and Processing Service to the authentication server. The default value is `iac-dtp`.
- **OAuth client secret:** Specifies the password for the single sign-on client for the Data Transformation and Processing Service client.

Specify the following details for **Workflow Service**:

- **OAuth Client ID:** Specifies the name that you want to use to identify the single sign-on client for Workflow Service to the authentication server. The default value is `wf`.
- **OAuth client secret:** Specifies the password for the single sign-on client for the Workflow Service client.

17 (Conditional) If you are installing Identity Governance for the first time, specify the following details during the Identity Governance installation:

- For the authentication server details, specify the Identity Manager OSP server details.
- Specify the LDAP user details when prompted for the Bootstrap administrator details.

NOTE: Based on the Identity Governance version that you are using (3.6 or 3.5.x), you can choose whether LDAP user is required or not. For more information, see Step 13 and Step 14.

---

18 (Conditional) If you have already installed Identity Governance, log in to the server where Identity Governance is installed and update the configuration values in the configuration utility (`configutil`) and configuration update utility (`configupdate`). For more information, see Step 3 and Step 4 of the "Using the Same Authentication Server as Identity Manager" on page 570 section.

19 Delete the `localhost` folder in the `tomcat/work/Catalina` directory.

20 Delete all the files and folders in the `/opt/netiq/idm/apps/tomcat/temp` directory.

21 Restart Tomcat on the Identity Governance server.

```
systemctl restart identity_tomcat.service
```

22 Restart Tomcat on the Identity Applications server.

```
systemctl restart netiq-tomcat.service
```

Configuring Session-timeout for Identity Manager and Identity Governance Integrated Setup

In Identity Manager and Identity Governance integrated setup, both Identity Applications and Identity Governance are browser-based applications, each having a distinct session time out property and value. The value represents the amount of time users can leave a page unattended in their web browser before the server displays a session-time-out warning.

The Identity Applications session time out value is represented by the `com.netiq.idm.session-timeout` property, whereas the Identity Governance session time out is represented by the `com.netiq.idm.osp.oauth.public.refreshTokenTTL` property. You must configure value for both these properties in the `ism-configuration.properties` file after Identity Manager and Identity Governance integration.

To set the session time out, perform the following actions:

1. Login in to the Identity Applications server.
2. Navigate to the `/opt/netiq/idm/apps/tomcat/conf/` location.
3. Open the `ism-configuration.properties` file in a text editor and add the two properties namely, `com.netiq.idm.session-timeout` and `com.netiq.idm.osp.oauth.public.refreshTokenTTL` with appropriate values.
4. Save the file and restart the Tomcat service.

NOTE: To keep the session-timeout warning same for both the applications, it is recommended that the value you set for `com.netiq.idm.osp.oauth.public.refreshTokenTTL` must be 120 seconds less than the value provided for the `com.netiq.idm.session-timeout` property.

---

38 Verifying Single Sign-on Access for the Identity Applications

After you install the identity applications and configure the settings for single sign-on, you should verify that you can log in to the individual applications and switch among them without logging out. By default, the applications use the following suffix in the URL link:

- Identity Manager Administration: `/idmadmin`
- Identity Manager Dashboard: `/idmdash`
- Identity Reporting: `/IDMRPT`

To customize the suffix, use the Configupdate utility. For more information, see Configuring the Settings for the Identity Applications in the *NetIQ Identity Manager Setup Guide for Linux* or Configuring the Settings for the Identity Applications in the *NetIQ Identity Manager Setup Guide for Windows*.

To verify single sign-on functionality:

1. In a new browser window on your identity applications server, enter the URL for the Dashboard:

   `https://server:port/idmdash`

   Do not log in to the Dashboard.
2. In your browser, navigate to the User Application:

   `https://server:port/IDM-context`
3. Verify that the User Application displays the same login page as shown in Step 1.
4. Log in to the User Application.
5. In the top right corner, click the **Home** icon and verify that you can access the Dashboard without logging on again.

39 Using SSL for Secure Communication

The identity applications and Identity Reporting use HTML forms for authentication. As a result, the login process might expose user credentials. NetIQ recommends that you enable SSL protocol to protect sensitive information. SSL protocol ensures that all communication between Identity Manager components is secured.

You must have certificates to configure Tomcat server to communicate using SSL. You can obtain certificates in two ways:

- External trusted Certificate Authority (CA) issued certificate
- Self-signed certificate

On Linux, the installation program automatically configures Identity Applications and Identity Reporting components with a secured connection (HTTPS) using the certificate issued by the Identity Vault. For a production environment, you are recommended to use a certificate issued by an external Certificate Authority. You need to manually configure a secured connection on Windows.

Checklist for Ensuring SSL Connections

To ensure secure connections among the identity applications components and Identity Reporting, NetIQ recommends that you perform the steps in the following checklist:

	Checklist Items
❑	1. Ensure that you have a keystore to store the authentication certificates. For more information, see “Creating a Keystore and Certificate Signing Request” on page 582.
❑	2. (Conditional) In a test environment, use self-signed certificates. For more information, see “Enabling SSL with a Self-signed Certificate” on page 585. For production environment, it is recommended to use external CA issued certificate.
❑	3. Ensure that you have configured the authentication server, identity applications, and Identity Reporting to support SSL communication. For more information, see “Updating the SSL Settings in the Configuration Utility” on page 592.
❑	4. Configure the authentication server, identity applications, and Identity Reporting to support SSL communication. For more information, see “Updating the SSL Settings for the Application Server” on page 590 and “Updating the SSL Settings in the Configuration Utility” on page 592.

Creating a Keystore and Certificate Signing Request

A keystore is a Java file that contains encryption keys and optionally, security certificates. To create a keystore, you can use the Java Keytool utility included in the JRE. You create the `.jks` file, generate a certificate into the keystore. Each certificate is associated with a unique alias. You place the keystore in the `conf` directory for your application server that supports the identity applications and Identity Reporting.

By default, the installer creates a keystore, namely `tomcat.ks`, in `/opt/netiq/idm/apps/tomcat/conf` and uses this keystore to configure the `https` connection. If you create a keystore file with the same name, replace this keystore file in this directory.

1. In a command prompt, navigate to the `conf` directory for your application server installation where you have deployed the identity applications. For example, `/opt/netiq/idm/apps/tomcat/conf` or `C:\NetIQ\idm\apps\tomcat\conf`.

   The `tomcat/conf` path is the default for the identity applications installed on Tomcat. The path can vary, depending on how you installed the application and Tomcat.

2. Set the environment path for creating the keystore using the following command:

   ```
   cd /opt/netiq/idm/apps/tomcat/conf
   export PATH=/opt/netiq/common/jre/bin:$PATH
   ```

   or

   ```
   cd C:\NetIQ\idm\apps\tomcat\conf
   set PATH=C:\NetIQ\idm\apps\jre\bin:%PATH%
   ```

3. Create the keystore using the following command:

   ```
   keytool -genkey -alias alias_name -keyalg RSA -keystore keystore_name -
   validity 3650 -keysize 2048
   ```

   For example:

   ```
   keytool -genkey -alias IDMkey -keyalg RSA -keystore tomcat.ks -validity
   3650 -keysize 2048
   ```

4. At the prompt, specify the parameter values according to the following considerations:
   - For first and last name, specify the fully qualified name of the server. For example:

     ```
     MyTomcatServer.NetIQ.com
     ```

   - Use correct spelling. If you spell any words incorrectly, you will see errors when you generate your signed certificate from the signing authority.

5. (Optional) Create a simple text file to save a copy of the information that you provide for the parameter values.

   Saving this information helps ensure that you supply the same information when you apply to the signing authority and when you import your certificate.

6. Copy the keystore file to the `/tomcat/conf` directory for each application server instance where you have deployed Identity Manager components and SSPR.

NOTE: In Linux system, you should change the owner of the file to novlua:

```
chown novlua:novlua [keystore_name.keystore]
```

7 To generate the CA certificate request, complete the following steps:

7a In the `conf` directory, create a simple text file named `your_request.csr`. For example, `IDMcertrequest.csr`.

7b Run the following command:

```
keytool -certreq -v -alias alias_name -file your_request.csr -
keypass keystore_password -keystore keystore_name -storepass
your_password
```

For example,

```
keytool -certreq -v -alias IDMkey -file IDMcertrequest.csr -keypass
IDMkeypass -keystore tomcat.ks -storepass IDMpass
```

When you run the command, the Keytool utility populates the .csr file with the appropriate data for requesting a certificate.

8 (Conditional) To obtain a signed certificate, submit the .csr file to a valid Certificate Authority.

9 Copy the certificate in the configuration directory of your application server.

For example, `/opt/netiq/idm/apps/tomcat/conf` or `C:\NetIQ\idm\apps\tomcat\conf`.

10 Stop Tomcat.

After creating a keystore and generating CA certificate request, perform the following procedures to import the certificates into the keystore:

- For external CA signed certificate, see "Enabling SSL with a External CA Signed Certificate" on page 583.
- For self-signed certificate, see "Enabling SSL with a Self-signed Certificate" on page 585.

Enabling SSL with a External CA Signed Certificate

For a production environment, use a signed certificate issued by a valid Certificate Authority. This section explains how to import a signed certificate into the default Tomcat application server for the identity applications.

This procedure assumes that you have a signed certificate from a valid Certificate Authority. For more information, see "Creating a Keystore and Certificate Signing Request" on page 582.

To use a signed certificate and SSL:

1 Copy the certificate in the configuration directory of your application server.

Linux: `/opt/netiq/idm/apps/tomcat/conf`

Windows: `C:\NetIQ\idm\apps\tomcat\conf`

2 To convert the root certificate to DER format, complete the following steps:

2a Double-click on your certificate stored in the `conf` directory.

2b In the Certificate dialog, click **Certificate Path**.

2c Select the root certificate that you received from the signing authority.

2d Click **View Certificate**.

2e Click **Details > copy to file**.

2f In the Export Certificate Wizard, click **next**.

2g Select **DER encoded binary for X.509 (.CER)** and then click **next**.

2h Create a new file to store the newly formatted certificate and store it in the `conf` directory for your application server.

2i Click **Finish**.

3 To import the converted certificate, complete the following steps:

3a In a command prompt, navigate to the `conf` directory for your application server. For example, `/opt/netiq/idm/apps/tomcat/conf` or `C:\NetIQ\idm\apps\tomcat\conf`.

3b Enter the following command:

```
keytool -import -trustcacerts -alias root -keystore keystore_name -file yourRootCA.der
```

For example:

```
keytool -import -trustcacerts -alias root -keystore tomcat.ks -file IDMTESTREE.der
```

NOTE: You must specify **root** as your alias.

After importing the certificate, the server displays **Certificate was added to keystore**.

3c Verify that the signed certificate is imported correctly into the `conf` directory using the following command:

```
keytool -list -v -alias root -keystore keystore_name
```

For example:

```
keytool -list -v -alias root -keystore tomcat.ks
```

The server lists your certificates.

4 NetIQ recommends you to import the signed certificates to `idm.jks`. This is a centralized keystore that stores all the certificates used by the identity applications and Identity Reporting. For example:

Linux: `keytool -import -trustcacerts -alias root -keystore /opt/netiq/idm/apps/tomcat/conf/idm.jks -file IDMTESTREE.der`

Windows: `keytool -import -trustcacerts -alias root -keystore C:\NetIQ\idm\jre\lib\security\cacerts -file IDMTESTREE.der`

5 Update the SSL settings for the application server, see "Updating the SSL Settings for the Application Server" on page 590.

6 Update the SSL settings in the Configuration utility. For more information, see "Updating the SSL Settings in the Configuration Utility" on page 592.

7 Update the SSL settings for Self Service Password Reset. For more information, see "Updating the SSL Settings for Self Service Password Reset" on page 593

8 Restart Tomcat.

Enabling SSL with a Self-signed Certificate

If you want to use a self-signed certificate in your test environment, since this type of certificate is easier to obtain than a signed certificate from a valid authority.

- "Exporting the Certificate Authority" on page 585
- "Generating the Self-signed Certificate" on page 586

Exporting the Certificate Authority

You can use iManager to export the Certificate Authority (CA) from your eDirectory server to generate your self-signed certificate.

1. Log in to iManager with the eDirectory administrator's username and password.
2. Click **Administration > Modify Object**.
3. In the Security container, browse to the CA object called *TreeName* `CA.Security`. For example, `IDMTESTTREE CA.Security`.
4. Click **OK**.
5. Click **Certificates > Self-Signed Certificate**.
6. Select the self-signed certificate that you want to use.

   Example: **Self Signed Certificate RSA**

   6a. Check **Self Signed Certificate RSA**.

   6b. Click **Validate**.
7. Click **Export**.
8. Clear **Export private key**.
9. Click **Export format > DER**.
10. Click **Next**.
11. Click **Save the exported certificate**.
12. Click **Save File**.

    iManager saves the file as *TreeName* `cert.der`. For example, `IDMTESTREE cert.der`.
13. Click **Close**.
14. Copy the certificate in the configuration directory of your application server (`cert.der`).

    For example, `/opt/netiq/idm/apps/tomcat/conf` or `C:\NetIQ\idm\apps\tomcat\conf`.
15. To import the root certificate, complete the following steps:

    15a. In a command prompt, navigate to the conf directory for your application server using following command:

    ```
    keytool -import -trustcacerts -alias root -keystore <keystore
    file>.keystore -file exported_certificate_filename.der
    ```

    Example:

    ```
    keytool -import -trustcacerts -alias root -keystore tomcat.ks -file
    cert.der
    ```

NOTE: You must specify **root** as your alias.

After importing the certificate, the server displays **Certificate was added to keystore**.

15b NetIQ recommends you to import root certificate to Java cacerts location also.

For example:

```
keytool -import -trustcacerts -alias root -keystore /opt/netiq/
common/jre/lib/security/cacerts -file cert.der
```

or

```
keytool -import -trustcacerts -alias root -keystore
C:\NetIQ\idm\jre\lib\security\cacerts -file cert.der
```

15c Verify the signed certificate is imported correctly in the `conf` directory by using following command:

```
keytool -list -v -alias root -keystore keystore_name
```

For example,

```
keytool -list -v -alias root -keystore tomcat.ks
```

The server lists the certificates.

Generating the Self-signed Certificate

Before generating the self-signed certificate, ensure that you have a keystore and certificate request file. For more information see "Creating a Keystore and Certificate Signing Request" on page 582.

1. Log in to iManager.
2. Navigate to **Certificate Server > Issue Certificate**.
3. Browse to the `.csr` file created in Step 7 in the "Creating a Keystore and Certificate Signing Request" on page 582.

   Example: `IDMcertrequest.csr`
4. Click **Next** twice.
5. For the certificate type, click **Unspecified**.
6. Click **Next** twice.

   iManager saves the file as `csr_request_name.der`. Example: `IDMcertrequest.der`
7. Copy the certificate in the configuration directory of your application server (`IDMcertrequest.der`).

   For example, `/opt/netiq/idm/apps/tomcat/conf` or `C:\NetIQ\idm\apps\tomcat\conf`.
8. To import the generated self-signed certificate, complete the following steps:

   8a In a command prompt, navigate to the `conf` directory for your application server using following command:

   ```
   keytool -import -alias alias_name -keystore <keystore_file> -file
   <signed_certificate_filename>.der
   ```

For example,

```
keytool -import -alias IDMkey -keystore tomcat.ks -file
IDMcertrequest.der
```

NOTE: You must specify the keystore name as your alias.

After importing the certificate, the server displays **Certificate was added to keystore**.

8b NetIQ recommends that you also import the self-signed certificate to the Java cacerts location.

For example:

```
keytool -import -alias IDMkey -keystore
/opt/netiq/common/jre/lib/security/cacerts -file IDMcertrequest.der
```

or

```
keytool -import -alias IDMkey -keystore
C:\NetIQ\idm\jre\lib\security\cacerts -file IDMcertrequest.der
```

8c Verify the signed certificate is imported correctly in the `conf` directory using the following command:

```
keytool -list -v -alias alias_name -keystore keystore_name
```

For example,

```
keytool -list -v -alias IDMkey -keystore tomcat.ks
```

The server lists the certificates.

9 Update the SSL settings for the Application server. For more information, see "Updating the SSL Settings for the Application Server" on page 590.

10 Update the SSL settings in the Configuration utility. For more information, see "Updating the SSL Settings in the Configuration Utility" on page 592.

11 Update the SSL settings for Self Service Password Reset. For more information, see "Updating the SSL Settings for Self Service Password Reset" on page 593

12 Restart Tomcat.

Enabling SSL Between Sentinel and Identity Manager Components

You can create and export a self-signed server certificate to ensure secure communication between Sentinel and Identity Manager components. Use a signed certificate issued by a valid Certificate Authority.

- "Enabling SSL between Sentinel and Identity Manager Engine/Remote Loader" on page 588
- "Enabling SSL between Sentinel and User Application" on page 589

Enabling SSL between Sentinel and Identity Manager Engine/ Remote Loader

1 To create a new certificate, complete the following steps:

   1a Log in to iManager.

   1b Click **NetIQ Certificate Server** > **Create Server Certificate**.

   1c Select the appropriate server.

   1d Specify a nickname for the server.

   1e Accept the rest of the certificate defaults.

2 To export the server certificate to `.pfx` format, complete the following steps:

   2a In iManager, select **Directory Administration** > **Modify Object**.

   2b Browse to and select the Key Material Object (KMO) object.

   2c Click **Certificates** > **Export**.

   2d Specify a password.

   2e Save the server certificate as a PKCS#12. For example, `certificate.pfx`.

3 Extract the private key from the exported certificate to `dxipkey.pem` file using the following command.

```
openssl pkcs12 -in certificate.pfx -nocerts -out dxipkey.pem –nodes
```

4 Extract the certificate to `dxicert.pem` file.

```
openssl pkcs12 -in certificate.pfx -nokeys -out dxicert.pem
```

5 To export the CA certificate of the eDirectory server created in Step 1 to Base64 format, complete the following steps:

   5a In iManager, navigate to **Roles and Tasks** > **NetIQ Certificate Access** > **User Certificates**.

   5b Browse and select the created certificate.

   5c Click **Export**.

   5d Select the **CA Certificate** as **OU=organizationCA.O=TREENAME** from the drop-down menu.

   5e Select the **Export Format** as **BASE64** from the drop-down menu.

   5f Click **Next** and save the certificate. For example, `cacert.b64`.

6 Import the CA certificate to a keystore using the following command:

```
keytool -import -alias <alias name> -file <b64 file> -keystore
<keystore file> –noprompt
```

   For example,

```
keytool -import -alias trustedroot -file cacert.b64 -keystore
idmKeystore.ks –noprompt
```

7 To import the certificate into the trust store of Audit Connector, complete the following steps:

   7a Log in to the Sentinel Main interface as an administrator.

   7b In the main ESM display, locate the Audit Server.

   7c Right-click the **Audit Server**, then click **Edit**.

   7d In the Security tab, select **Strict**.

NOTE: By default, it is configured to use **Open** (insecure) mode to allow initial connectivity. However, when you are using it in a production environment, ensure that you set the mode to **Strict**.

7e Click **Import** and navigate to the certificate you created in Step 6. For example, `idmkeystore.ks`.

7f Click **Open** and then click **Save**.

7g Restart Audit Server.

8 Restart Identity Manager services.

Enabling SSL between Sentinel and User Application

1 To create a new certificate, complete the following steps:

1a Log in to iManager.

1b Click **NetIQ Certificate Server** > **Create User Certificate**.

1c Select the appropriate user.

1d Specify a nickname for the user.

1e In **Creation Method**, select **Custom**.

1f Accept the rest of the certificate defaults.

1g Click **Next**.

1h In **Custom Extensions**, Select **New DER Encoded Extensions**.

1i Browse to `\products\RBPM\ext.der` custom extension.

1j (Optional) Specify the e-mail address.

1k Review the certificate parameters and click **Finish**.

2 To export the user certificate, complete the following steps:

2a Click **NetIQ Certificate Access > User Certificates**.

2b Select the user certificate that is imported in Step 1.

2c Select the valid user certificate and click **Export**.

2d Specify a password.

2e Save the user certificate as a PKCS12. For example, `certificate.pfx`.

3 Extract the private key from the exported certificate to `key.pem` file using the following command.

```
openssl pkcs12 -in certificate.pfx -nocerts -out key.pem –nodes
```

4 Extract the certificate to `cert.pem` file.

```
openssl pkcs12 -in certificate.pfx -nokeys -out cert.pem
```

5 Stop User Application.

6 Add the private key and certificate to the configupdate utility.

6a Open the configupdate utility.

6b Click **Show Advanced Options**.

6c In the **NetIQ Sentinel Digital Signature Certificate** field, copy the `cert.pem`.

6d In the **NetIQ Sentinel Digital Signature Private Key** field, navigate to the location where you have extracted the private key (`key.pem`) and import the key.

6e Save the changes to the configupdate utility.

7 Restart Identity Applications.

8 To export the CA certificate of the eDirectory server created in Step 1 to Base64 format, complete the following steps:

8a In iManager, navigate to **Roles and Tasks** > **NetIQ Certificate Access** > **User Certificates**.

8b Select the created certificate.

8c Click **Export** and clear the Export private key check box.

8d Select the **Export Format** as **BASE64** from the drop-down menu.

8e Click **Next** and save the certificate. For example, `cacert.b64`.

9 Import the CA certificate to a keystore using the following command:

```
keytool -import -alias <alias name> -file cacert.b64 -keystore
<keystore file> –noprompt
```

For example,

```
keytool -import -alias trustedroot -file cacert.b64 -keystore
idmKeystore.ks –noprompt
```

10 To import the certificate into the trust store of Audit Connector, complete the following steps:

10a Log in to the Sentinel Main interface as an administrator.

10b In the main ESM display, locate the Audit Server.

10c Right-click the **Audit Server**, then click **Edit**.

10d In the **Security** tab, select **Strict**.

NOTE: By default, it is configured to use **Open** (insecure) mode to allow initial connectivity. However, when you are using it in a production environment, ensure that you set the mode to **Strict**.

10e Click **Import** and navigate to the certificate you created in Step 9. For example, `idmKeystore.ks`.

10f Click **Open** and then click **Save**.

10g Restart Audit Server.

11 Restart Identity Applications.

Updating the SSL Settings for the Application Server

The installer automatically configures the application server that hosts the identity applications and Identity Reporting to support SSL communication. It creates the connector by default in `server.xml` file located by default in the following paths:

Linux: `/opt/netiq/idm/apps/tomcat/conf/`

Windows: `C:\NetIQ\idm\apps\tomcat\conf`

```
<Connector port="https_port"
protocol="org.apache.coyote.http11.Http11NioProtocol" maxThreads="150"
SSLEnabled="true" scheme="https" secure="true" clientAuth="false"
sslProtocol="TLSv1.2" keystoreFile="path_to_keystore_file"
keystorePass="keystore_password" sslEnabledProtocols="TLSv1.2" />
```

where:

keystoreFile

Specifies the path to the keystore file, for example, `idmapps.keystore` file. Place the file in the respective directories based on your platform:

Linux: `/opt/netiq/idm/apps/tomcat/conf/`

Windows: `C:\NetIQ\idm\apps\tomcat\conf`

keystorePass

Specifies the password for the `idmapps.keystore` file.

You must verify that the keystore password and the keystore file path are correct in `server.xml` file.

To modify the values supplied by the installation, perform the following actions:

1. Stop Tomcat.

   ```
   systemctl stop netiq-tomcat.service
   ```

2. Navigate to the `conf` directory for Tomcat, located by default in the following directories:

   Linux: `/opt/netiq/idm/apps/tomcat/conf/`

   Windows: `C:\NetIQ\idm\apps\tomcat\conf`

3. Ensure that you have a keystore file in the `conf` directory. For example, `tomcat.ks` on Linux or `idmapps.keystore` on Windows.

   If you create the keystore file after performing this procedure, ensure that you use the same file name that you provide in this procedure. For more information, see "Creating a Keystore and Certificate Signing Request" on page 582.

4. In a text editor, open the `server.xml` file from the `conf` directory.

5. Configure SSL port for the Tomcat server.

   For example, connector port for SSL is 8543.

   Also, update the `redirectPort` attribute to 8543 and save `server.xml`.

   For example:

   ```
   <Connector port="https_port"
   protocol="org.apache.coyote.http11.Http11NioProtocol" maxThreads="150"
   SSLEnabled="true" scheme="https" secure="true" clientAuth="false"
   sslProtocol="TLSv1.2" keystoreFile="path_to_keystore_file"
   keystorePass="keystore_password" sslEnabledProtocols="TLSv1.2" />
   ```

6. Start Tomcat.

   ```
   systemctl start netiq-tomcat.service
   ```

Updating the SSL Settings in the Configuration Utility

On Linux, the installer automatically configures the SSL settings. If required, you can modify the values using the configuration utility. Ensure that Tomcat is stopped.

```
systemctl stop netiq-tomcat.service
```

On Windows, when you install the identity applications and Identity Reporting, you should specify *https* for the communication method. For example, **Protocol**. However, after installation, you can use the ConfigUpdate utility to ensure that the applications communicate with SSL. For more information about these parameters, see Configuring the Settings for the Identity Applications in the *NetIQ Identity Manager Setup Guide for Linux* or Configuring the Settings for the Identity Applications in the *NetIQ Identity Manager Setup Guide for Windows*.

To update the SSL settings:

1. Stop Tomcat.
2. Navigate to the RBPM Configuration utility, located by default in the installation directory for the identity applications.

   Linux: `/opt/netiq/idm/apps/configupdate`

   Windows: `C:\NetIQ\idm\apps\UserApplication`
3. At the command prompt, use one of the following methods to run the configuration utility:
   - **Linux**: `./configupdate.sh`
   - **Windows**: `configupdate.bat`

   NOTE: You might need to wait a few minutes for the utility to start up.

   ---
4. (Conditional) If you configure SSL in the configupdate utility, navigate to the **Authentication** tab and replace all the references mentioned in the **SSO Clients** tab.

   `https://<IP address>:<SSL Port number>`

   For example,

   `https://192.168.0.1:8543`
5. Click **Authentication > Show Advanced Options**, and then modify the following settings:

   OAuth server TCP port

   Specifies the port for the authentication server.

   OAuth server is using TLS/SSL

   Specifies that you want the authentication server to use TLS/SSL protocol for communication.

   Optional TLS/SSL keystore file

   Specifies the path and filename of the Java JKS keystore file that contains the authentication server trust certificate. This parameter applies when the authentication server uses TLS/SSL protocol, and the trust certificate for the authentication server is not in the JRE trust store (`cacerts`).

   Optional TLS/SSL keystore password

   Specifies the password used to load the keystore file for the TLS/SSL authentication server.

OAuth keystore file

Specifies the path to the Java JKS keystore file you want to use for authentication. The keystore file must contain at least one public/private key pair.

OAuth keystore file password

Specifies the password used to load the OAuth keystore file.

Key alias of key for use by OAuth

Specifies the name of the public/private key pair in the OSP keystore file that you want to use to symmetric key generation.

Key password key for use by OAuth

Specifies the password for the private key used by the authentication server.

6 Click **SSO Clients**.

7 Update all of the URL settings, such as **URL link to landing page** and **OAuth redirect URL**.

These settings specify the absolute URL to which the authentication server directs a browser client when authentication is complete.

Use the following format: `https://DNS_name:sslport/path`. For example, `https://nqserver.testsite:8543/landing/com.netiq.test`.

8 Save the changes in the configuration utility.

Updating the SSL Settings for Self Service Password Reset

To modify the SSL settings for SSPR, you must be logged in to the application.

1 In a browser, enter the `https` URL that you specified in the Configuration utility for the Landing page. For example, `https://myserver.host:8543/sspr`.

2 Log in using administrator credentials.

3 Click on user on the top-right corner and then click **Configuration Editor**.

4 Specify the configuration password and click **Sign In**.

5 Click **Settings** > **Application** > **Application** and ensure that all URLs use the HTTPS protocol and correct ports.

6 Click **Settings** > **Security** > **Web Security** and ensure that the **Redirect Whitelist** uses HTTPS protocol and correct port.

7 Click **Settings** > **Single Sign On (SSO) Client** > **OAuth** and ensure that all URLs use the HTTPS protocol and correct ports. Also, ensure that the correct certificate exists for the **OAUTH Web Service Certificate**. If the certificate is not correct, click **Clear** and then click **Import from Server** to import a new certificate.

8 Click  at the top-right corner.

Troubleshooting Tip

After updating the SSL settings for SSPR, if you are not able to access the SSPR landing page, perform the following actions to update the necessary URLs in the SSPR`Configuration.xml` file.

1 Navigate to the SSPR`Configuration.xml` file:

`/opt/netiq/idm/apps/sspr/sspr_data`

2 Update all the URLs with appropriate IP address and port numbers.

`https://<IP address>:<SSL Port number>`

Example:

`https://192.168.0.1:8543`

40 REST Services

The identity applications components incorporate several REST APIs that enable different features within Identity Applications. The REST APIs and the corresponding documentation are available in the `idmappsdoc.war` file. The `war` is automatically deployed when Identity Applications are installed.

The APIs are broadly classified into three categories:

Access

This category includes the APIs that are related to administrator settings, user navigation rights, user permissions, and assignment details.

Admin

This category includes the APIs that are related to logging, caching provisioning display settings, and administrator assignments.

Catalog

This category includes the APIs that are related to roles, resources, SoDs, and driver services.

For more information about REST services, access `idmappsdoc` in your identity applications server.

NOTE: Identity applications REST services use OAUTH2 protocol to provide authentication. You can invoke these APIs using a browser or curl command in scripts to automate the administrative tasks. For more information, see the *REST API* documentation.

To access the REST API documentation on the server where identity applications are installed, specify `https://<identity applications servername>:<Port>/idmappsdoc`, in the address bar of your browser. For example: `https://192.168.0.1:8543/idmappsdoc`. Alternatively, access this documentation from REST API Reference (https://www.netiq.com/documentation/identity-manager-developer/rest-api-documentation/idmappsdoc/index.html).

Use Cases for Identity Applications REST API

This topic describes some of the basic use cases that you can accomplish using Identity Applications REST API.

Before You Begin

Identity Applications provide a complete API accessible via HTTP/HTTPS. See the REST API documentation for detailed information about the available endpoints. This information will also help you to make the API calls described in this topic.

Use Cases

The following use case describe some of the basic operations that can be performed by using Identity Applications REST APIs:

Role Management

Roles are a key component in role-based provisioning and role-based access control. A role represents a set of permissions that allows a user to perform defined activities. A role can be mapped to one or more roles, resources, and entitlements from different connected systems. You can assign any role to any user in your organization.

The following examples describe some of the basic role management operations. The role and resource administration user interface of Identity Applications leverages the same APIs to accomplish the role management tasks.

Access Rights Needed: Role Administrator or Role Manager

Example 1: Create a Role

Build roles depending on your organization's needs to efficiently provision users with the resources that they need access to, such as a software application or a security access card.

A role is created with a specific role level. You can associate a role with other roles, map it with organizational resources, and assign it to users, groups, or containers. Use the following example endpoint to create a role named PhysicsProfessor1.

Request URL: https://IP:port/IDMProv/rest/catalog/roles

Request Method: POST

Sample Request Payload:
```
{"id": "PhysicsProfessor1", "name":
"PhysicsProfessor1",  "localizedNames": [{"locale": "en", "name":
"PhysicsProfessor1" }], "description": "PhysicsProfessor1",
"localizedDescriptions": [{"locale": "en", "desc":  "PhysicsProfessor1"}],
"categories": [], "owners": [], "level": 30,  "approvalRequired": false,
"revokeRequired": false}
```

Sample Success Response: Returns a JSON output containing the details of the role.

```
{"id":
"cn=PhysicsProfessor1,cn=Level30,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn
=User Application Driver,cn=driverset1,o=system", "name":
"PhysicsProfessor1",  "description": "PhysicsProfessor1", "categories":
[null], "owners": [null],  "level": 30, "approvalRequired": false,
"revokeRequired": false,  "localizedNames": [{"locale": "en", "name":
"PhysicsProfessor1" }],  "localizedDescriptions": [{"locale": "en",
"desc": "PhysicsProfessor1"}]}
```

Example 2: Modify a Role

You can modify all role parameters except Level and Subcontainer. You can edit each role separately or multiple roles at the same time. For example, if you want to set the approval for a role, use the following example endpoint:

Request URL: https://IP:port/IDMProv/rest/catalog/roles/role

Request Method: PUT

Sample Request Payload:

```
{"roles": [{"id":
"cn=PhysicsProfessor1,cn=level30,cn=roledefs,cn=roleconfig,cn=appconfig,cn
=user application driver,cn=driverset1,o=system",  "name":
"PhysicsProfessor1",  "localizedNames": [{ "locale": "en",  "name":
"PhysicsProfessor1"}],  "description": "PhysicsProfessor1",
"localizedDescriptions": [{ "locale": "en",  "desc":  "PhysicsProfessor1"
}], "categories": [{"id": "system",  "name": "System Roles", "type":
"category" }], "owners": [],  "approvalRequired": true,
"approvalIsStandard": true,  "approvalApprovers": [{ id":
"cn=kevin,o=data",  "name": "kevin klaus",  "type": "user", "sequence":
0}]}]}
```


Sample Success Response: Returns a JSON output containing the details of the updated role.


```
{"success": "true",  "succeeded": {"id":
"cn=PhysicsProfessor1,cn=level30,cn=roledefs,cn=roleconfig,cn=appconfig,cn
=user application driver,cn=driverset1,o=system"}}
```


Example 3: Delete a Role

You can delete a role when it is no longer required in the system. When you issue the endpoint to delete a role, Identity Applications set the status of the role to a pending delete status. The Role and Resource Service driver notes the change of status for the role and removes the corresponding references and then deletes the role from the Identity Vault.

Request URL: https://IP:port/IDMProv/rest/catalog/roles/role

Request Method: DELETE

Sample Request Payload:


```
{"roles":[{"id":"cn=PhysicsProfessor1,cn=level30,cn=roledefs,cn=roleconfig
,cn=appconfig,cn=user application
driver,cn=driverset1,o=system","name":"PhysicsProfessor1","description":"P
hysicsProfessor1","level":30,"roleLevel":"Business
Role","rowId":"cn=PhysicsProfessor1,cn=level30,cn=roledefs,cn=roleconfig,c
n=appconfig,cn=user application driver,cn=driverset1,o=system"}]}
```


Sample Success Response: Returns a JSON output containing the DN of the deleted role.


```
{"success":"true","succeeded":{"id":"cn=PhysicsProfessor1,cn=level30,cn=ro
ledefs,cn=roleconfig,cn=appconfig,cn=user application
driver,cn=driverset1,o=system"}}
```


Example 4: Obtain the Details of a Role

A role can be associated with other roles, mapped with organizational resources, and assigned to users, groups, or containers. To obtain the details of a role, use the following example endpoint:

Request URL: https://IP:port/IDMProv/rest/catalog/roles/role

Request Method: POST

Sample Request Payload: {"roles": {"id":
"cn=PhysicsProfessor1,cn=level30,cn=roledefs,cn=roleconfig,cn=appconfig,cn
=user application driver,cn=driverset1,o=system"}}
{"arraySize":1,"roles":[{"id":"cn=PhysicsProfessor1,cn=level30,cn=roledefs
,cn=roleconfig,cn=appconfig,cn=user application
driver,cn=driverset1,o=system","name":"PhysicsProfessor1","description":"P
hysicsProfessor1","approvalRequestDef":"cn=Role
Approval,cn=RequestDefs,cn=AppConfig,cn=User Application
Driver,cn=driverset1,o=system","approvalApprovers":[{"id":"cn=kevin,o=data
","name":"kevin
kevin","type":"user","sequence":"0"}],"level":30,"roleLevel":{"name":"Busi
ness
Role","level":30,"cn":"cn=Level30,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,c
n=User Application
Driver,cn=driverset1,o=system"},"subContainer":"","approvalIsStandard":tru
e,"approvalRequired":true,"localizedNames":[{"locale":"en","name":"Physics
Professor1"}],"localizedDescriptions":[{"locale":"en","desc":"PhysicsProfe
ssor1"}]}]}

Sample Success Response: Returns the details of the role.

{"roles": {"id":
"cn=PhysicsProfessor1,cn=level30,cn=roledefs,cn=roleconfig,cn=appconfig,cn
=user application driver,cn=driverset1,o=system"}}
{"arraySize":1,"roles":[{"id":"cn=PhysicsProfessor1,cn=level30,cn=roledefs
,cn=roleconfig,cn=appconfig,cn=user application
driver,cn=driverset1,o=system","name":"PhysicsProfessor1","description":"P
hysicsProfessor1","approvalRequestDef":"cn=Role
Approval,cn=RequestDefs,cn=AppConfig,cn=User Application
Driver,cn=driverset1,o=system","approvalApprovers":[{"id":"cn=kevin,o=data
","name":"kevin
kevin","type":"user","sequence":"0"}],"level":30,"roleLevel":{"name":"Busi
ness
Role","level":30,"cn":"cn=Level30,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,c
n=User Application
Driver,cn=driverset1,o=system"},"subContainer":"","approvalIsStandard":tru
e,"approvalRequired":true,"localizedNames":[{"locale":"en","name":"Physics
Professor1"}],"localizedDescriptions":[{"locale":"en","desc":"PhysicsProfe
ssor1"}]}]}

User Management

A user is an individual in an organization (typically an employee) who requires access to one or more resources. Users are added to Identity Manager to provision resources to them. When you create a user, the user is stored as a user object in the Identity Vault. Each user object is associated with attributes that act as identifiers for the user. For example, a user's attributes can include user's first name, last name, phone numbers, employee number, manager, and e-mail address.

The following examples describe some of the basic user management operations, such as create a user, delete an existing user, and update the details of an existing user such as change the email ID, phone number, and manager. You can edit a single user or multiple users at the same time.

Access Rights Needed: Provisioning Administrator

Example 1: Create a User

Use the following example endpoint to create a user.

Request URL: https://IP:port/IDMProv/rest/access/user

Request Method: POST

Sample Request Payload:

```
{"container":"o=data","password":"novell","attributes":[{"key":"FirstName"
,"dataType":"String","isRequired":true,"values":["kevin"],"action":"add"},
{"key":"LastName","dataType":"String","isRequired":true,"values":["kevin"]
,"action":"add"},{"key":"CN","dataType":"String","isRequired":false,"value
s":["kevin"],"action":"add"} ] }
```

Sample Success Response: Returns a JSON output containing the details of the user created in the Identity Vault.

```
{"dn":"cn=kevin,o=data"}
```

Example 2: Modify a User

You can modify all user attributes that are configured in the Directory Access Layer. For example, if you want to change the title of a user, use the following example endpoint:

Request URL: https://IP:port/IDMProv/rest/access/user

Request Method: PUT

Sample Request Payload:

```
{"dn":"cn=ABlake,o=data","attributes":[{"key":"Title","values":["SE"],"act
ion":"UPDATE"},{"key":"Email","values":[],"action":"UPDATE"},{"key":"Telep
honeNumber","values":[],"action":"UPDATE"},{"key":"manager","values":[],"a
ction":"UPDATE"},{"key":"directReports","values":[],"action":"UPDATE"}]}
```

Sample Success Response: Returns a JSON output containing all the details of the user.

```
{"dn":"cn=ABlake,o=data","attributes":[{"name":"Hidden attribute
List","key":"srvprvHideAttributes","hidden":false,"dataType":"String"},{"n
ame":"Email","key":"Email","
hidden":false,"dataType":"String"},{"name":"Manager","key":"manager","hidd
en":false,"dataType":"DN"},{"name":"Direct
Reports","key":"directReports","hidden":false,"dataType":"DN"},{"name":"Fi
rst
Name","key":"FirstName","values":["ABlake"],"hidden":false,"dataType":"Str
ing"},{"name":"Title","key":"Title","values":["SE"],"hidden":false,"dataTy
pe":"String"},{"name":"CN","key":"CN","values":["ABlake"],"hidden":false,"
dataType":"String"},{"name":"Telephone
Number","key":"TelephoneNumber","hidden":false,"dataType":"String"},{"name
":"User
Preferences","key":"srvprvUserPrefsPlus","hidden":false,"dataType":"Stream
"},{"name":"User
```

```
Photo","key":"UserPhoto","hidden":false,"dataType":"Binary"},{"name":"Depa
rtment","key":"Department","hidden":false,"dataType":"String"},{"name":"Qu
ery
List","key":"srvprvQueryList","hidden":false,"dataType":"String"},{"name":
"User
Preferences","key":"srvprvUserPrefs","hidden":false,"dataType":"String"},{
"name":"Preferred
Notification","key":"NotificationPrefs","hidden":false,"dataType":"String"
},{"name":"Last
Name","key":"LastName","values":["ABlake"],"hidden":false,"dataType":"Stri
ng"},{"name":"Dashboard
Preferences","key":"srvprvDashboardPrefs","hidden":false,"dataType":"Strea
m"},{"name":"Region","key":"Location","hidden":false,"dataType":"String"},
{"name":"Group","key":"group","hidden":false,"dataType":"DN"}]}
```


Example 3: Delete a User

You can delete or make a user inactive if the user is no longer part of the organization. To delete a user, use the following example endpoint:

Request URL: https://IP:port/IDMProv/rest/access/user

Request Method: DELETE

Sample Request Payload: `{"members":[{"dn":"cn=ABlake,o=data"}]}`

Sample Request Payload: Returns a JSON output containing the DN of the deleted user.


```
{"success":true,"succeeded":[{"id":"cn=ABlake,o=data","name":"Aston
Blake"}]}
```


Assign a Role to a User

An administrator can assign a role to a user, group, or a container.

To assign a role to a user, use the following example endpoint:

Access Rights Needed: Role Administrator or Role Manager

Request URL: https://IP:port/IDMProv/rest/catalog/roles/role/assignments/assign

Sample Request Method: POST

Sample Request Payload:

```
{"reason": "change of designation", "assignments":
[{"id":
"cn=rolea,cn=level30,cn=roledefs,cn=roleconfig,cn=appconfig,cn=user
application driver,cn=driverset1,o=system", "assignmentToList": [{
"assignedToDn": "cn=kevin,o=data", "subtype": "user"}], "effectiveDate":
"", "expiryDate": ""}]}
```


Sample Success Response: Returns a JSON output containing a list of role assignments of the user.


```
{"success":"true","succeeded":{"id":"cn=rolea,cn=level30,cn=roledefs,cn=ro
leconfig,cn=appconfig,cn=user application
driver,cn=driverset1,o=system","assignToUsers":{"dn":"cn=kevin,o=data","na
me":"kevin"}}}
```

The role catalog user interface of Identity Applications leverages the same API for assigning a role to a user.

Request a Role for Yourself

Users can request a role for themselves.

Access Rights Needed: Any end-user who can log in to the Dashboard can access it.

Request URL: https://IP:port/IDMProv/rest/access/requests/permissions

Request Method: POST

Sample Request Payload:

```
{"reason":"change of
designation","effDate":"","expDate":"","reqPermissions":[{"id":"cn=Physics
Professor1,cn=level30,cn=roledefs,cn=roleconfig,cn=appconfig,cn=user
application
driver,cn=driverset1,o=system","dn":"cn=PhysicsProfessor1,cn=Level30,cn=Ro
leDefs,cn=RoleConfig,cn=AppConfig,cn=User Application
Driver,cn=driverset1,o=system","name":"PhysicsProfessor1","desc":"PhysicsP
rofessor1","entityType":"role","bulkRequestable":true,"link":"/IDMProv/
rest/access/permissions/
item","multiAssignable":true,"edition":"rbpm.role.1552670228469","isNewFor
m":false,"isExpirationRequired":"false"}]}
```


Sample Success Response: Returns a JSON output containing the details of the requested role.


```
{"success":true,"succeeded":[{"id":"cn=PhysicsProfessor1,cn=level30,cn=rol
edefs,cn=roleconfig,cn=appconfig,cn=user application
driver,cn=driverset1,o=system","instanceId":"cn=20190320201141-
05e23fe7133c4014bb4bcefc2e9d980b-
0,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=User Application
Driver,cn=driverset1,o=system"}]}
```

41 Troubleshooting

The following sections contain information about troubleshooting different components of identity applications:

- "Using Log Files for Troubleshooting" on page 603
- "Troubleshooting E-Mail Based Approval Issues" on page 620
- "Troubleshooting Self Service Password Reset Issues" on page 621
- "Troubleshooting Authentication Issues" on page 622
- "Troubleshooting General Issues" on page 625
- "Troubleshooting Multi-Threaded Role and Resource Service driver Issues" on page 630

Using Log Files for Troubleshooting

The following sections provide information about how to use log files for troubleshooting problems:

- "Customizing Logging Settings" on page 603
- "Virtual Data Access Logging" on page 604
- "When a Code Map Refresh Is Triggered" on page 607
- "When Multiple Users Try to Authenticate From Different Interfaces" on page 608
- "When an E-Mail Approval Notification is Not Delivered" on page 609
- "When a Role Is Requested" on page 609
- "When a Role Is Listed in Role Catalog" on page 612
- "Schema Fails to Update When Updated Using a User Account That Was Not Used to Create the Schema" on page 614
- "Checking the Status of Database Schema Validation" on page 616
- "Determining if Liquibase Changeset Has Executed" on page 616
- "When Assigning a Resource to a User That Does Not Exist" on page 618
- "When Checking the Workflow Engine Heartbeat" on page 618
- "catalina.out File Does Not Rotate the Log on Linux" on page 619

Customizing Logging Settings

By default, log entries include only function names and not the complete class path that can make it difficult to determine which package is generating a particular message. To include the complete class path in a log message, change all the instances of the following entry in the `userapp-log4j.xml` file:

```
<param name="ConversionPattern" value="%d [%p] %c{1} %m%n"/>
```

to

```
<param name="ConversionPattern" value="%d [%p] %C %m%n"/>
```

After making this change, the following example trace entry:

```
2017-08-29 16:05:05,392 DEBUG [RBPM] Entity Definition found: sys-nrf-navitem
```

looks similar to this:

```
2017-08-29 16:05:05,392 DEBUG
com.novell.srvprv.impl.vdata.definition.VirtualDataDefinition- [RBPM]
Entity Definition found: sys-nrf-navitem
```

NOTE: The examples in the subsequent sections contain a complete class path to help you correctly interpret the meaning of log entries in the `catalina.out` file.

Virtual Data Access Logging

The Virtual Data Access (VDA) trace is logged to the `catalina.out` file when you look for an entity definition.

The VDA issues all of the identity applications LDAP queries for identity data such as entity definition and attributes on the Identity Vault. First it queries the information from the local cache residing on the identity applications server. If the information is not found in the local cache, it queries the Identity Vault. After locating the object, the identity applications read the attributes of the object and map the attributes with the entity definition in the local cache. When the entity definition matches, the identity applications send the data to the client. The trace looks similar to the following:

```
2017-08-29 16:05:05,389 [http-bio-8443-exec-10] DEBUG
com.novell.idm.nrf.util.CacheUtil- [RBPM] Role object was found in cache: cache-
key-nrf-config

2017-08-29 16:05:05,389 [http-bio-8443-exec-10] DEBUG
com.novell.idm.nrf.util.CacheUtil- [RBPM] Role object RETRIEVED from cache: cache-
key-nrf-config

2017-08-29 16:05:05,392 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM]
VDA.getEntityResultList

2017-08-29 16:05:05,392 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataModel- [RBPM]
VDM.getEntityDefinition(String, Locale):sys-nrf-navitem

2017-08-29 16:05:05,392 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.spi.vdata.util.CacheUtil- [RBPM] DAL Definition is null/not
found in cache: VDM_ENTITY_DEFINITION_sys-nrf-navitem, Is object in cache?false

2017-08-29 16:05:05,392 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.definition.VirtualDataDefinition- [RBPM] Entity
Definition found: sys-nrf-navitem

2017-08-29 16:05:05,393 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.spi.vdata.util.CacheUtil- [RBPM] DAL definition was found in
cache: VDD_ENTITY_ATTR_sys-nrf-navitem

2017-08-29 16:05:05,393 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.spi.vdata.util.CacheUtil- [RBPM] DAL Definition RETRIEVED from
cache: VDD_ENTITY_ATTR_sys-nrf-navitem
```

```
2017-08-29 16:05:05,393 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.spi.vdata.util.CacheUtil- [RBPM] PUT new DAL Definition into
cache: VDM_DEFINITION_ATTRIBUTE_LIST_sys-nrf-navitem

2017-08-29 16:05:05,394 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataModel- [RBPM]
VDM.getEntityDefinition(String, Locale):sys-nrf-navitem

2017-08-29 16:05:05,394 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM]
VDA.getEntityResultList query filter: (&(objectClass=nrfNavItem))

2017-08-29 16:05:05,439 [http-bio-8443-exec-10] DEBUG
com.novell.idm.security.ui.UIProvSecurityUtil- [RBPM] not Admin or Compliance tab,
so checkAccess with resource =
cn=WorkDashBoard,cn=NavItems,cn=UIConfig,cn=AppConfig,cn=UserApplication,cn=driver
set,ou=idm,ou=services,o=system

2017-08-29 16:05:05,443 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataModel- [RBPM]
VDM.getEntityDefinition(String, Locale):sys-nrf-user

2017-08-29 16:05:05,444 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.definition.VirtualDataDefinition- [RBPM] Entity
Definition found: sys-nrf-user

2017-08-29 16:05:05,444 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.spi.vdata.util.CacheUtil- [RBPM] PUT new DAL Definition into
cache: VDM_ENTITY_DEFINITION_sys-nrf-user

2017-08-29 16:05:05,444 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] VDA.getEntity:
cn=mytestuser,dc=data

2017-08-29 16:05:05,444 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.spi.vdata.util.CacheUtil- [RBPM] DAL definition was found in
cache: VDD_ENTITY_ATTR_sys-nrf-user

2017-08-29 16:05:05,444 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.spi.vdata.util.CacheUtil- [RBPM] DAL Definition RETRIEVED from
cache: VDD_ENTITY_ATTR_sys-nrf-user

2017-08-29 16:05:05,445 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.spi.vdata.util.CacheUtil- [RBPM] PUT new DAL Definition into
cache: VDM_DEFINITION_ATTRIBUTE_LIST_sys-nrf-user

2017-08-29 16:05:05,446 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] VDA.getLdapAttributes
Attributes and values

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] Attribute ID: mail

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM]
mytestuser@acme.com

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] Attribute ID:
modifyTimestamp

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] 20150716124158Z

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
```

```
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] Attribute ID:
givenName

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] mytestuser

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] Attribute ID:
objectClass

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] inetOrgPerson

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] organizationalPerson

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] Person

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] ndsLoginProperties

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] Top

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] srvprvEntityAux

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] Attribute ID: sn

2017-08-29 16:05:05,478 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] mytestuser

2017-08-29 16:05:05,479 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] Attribute ID: cn

2017-08-29 16:05:05,479 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] VDA.checking if
object instance contains the required objectClass per DAL definition: sys-nrf-user

2017-08-29 16:05:05,479 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] VDA.does contain
required (search=true or auxilliary=false) objectClass:inetOrgPerson

2017-08-29 16:05:05,479 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] VDA.object instance
is correct type

2017-08-29 16:05:05,479 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataModel- [RBPM]
VDM.getEntityDefinition(String, Locale):sys-nrf-user

2017-08-29 16:05:05,479 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.spi.vdata.util.CacheUtil- [RBPM] DAL Definition RETRIEVED from
cache: VDM_ENTITY_DEFINITION_sys-nrf-user

2017-08-29 16:05:05,479 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] VDA.getEntity:
cn=mytestuser,dc=data

2017-08-29 16:05:05,480 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.spi.vdata.util.CacheUtil- [RBPM] DAL Definition RETRIEVED from
cache: VDM_DEFINITION_ATTRIBUTE_LIST_sys-nrf-user
```

```
2017-08-29 16:05:05,481 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] VDA.getLdapAttributes
Attributes and values

2017-08-29 16:05:05,481 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM]   Attribute ID: mail

2017-08-29 16:05:05,481 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM]
mytestuser@acme.com

2017-08-29 16:05:05,481 [http-bio-8443-exec-10] DEBUG
com.novell.srvprv.impl.vdata.model.VirtualDataAccess- [RBPM] 20150716124158Z
```

When an entity model is changed, you must clear the VDA cache for the changes to take effect. For example, entity changes occur when a new attributes is added or the existing attributes are modified or removed.

When a Code Map Refresh Is Triggered

When you initiate a code map refresh cycle, sometimes the refresh cycle is not successful. When this occurs, the trace prints messages similar to the following:

```
2016-11-14 12:28:12,045 [Timer-1] INFO com.novell.idm.nrf.service.CodeMapEngine-
[RBPM] Building the Entitlement CODE MAP tables...

2016-11-14 12:28:12,450 [Timer-1] ERROR com.novell.idm.nrf.service.CodeMapEngine-
[RBPM] Unable to complete the CODE MAP refresh for entitlement:
cn=exchangemailbox,cn=ad,cn=dset,ou=idm,o=system.

2016-11-14 12:28:12,454 [Timer-1] INFO com.novell.idm.nrf.service.CodeMapEngine-
[RBPM] CODE MAP refresh on entitlement:
cn=exchangemailbox,cn=ad,cn=dset,ou=idm,o=system, processed, next refresh time:
20161115122812-0500.

2016-11-14 12:28:12,646 [Timer-1] ERROR com.novell.idm.nrf.service.CodeMapEngine-
[RBPM] Unable to complete the CODE MAP refresh for entitlement:
cn=group,cn=ad,cn=dset,ou=idm,o=system.

2016-11-14 12:28:12,835 [Timer-1] ERROR com.novell.idm.nrf.service.CodeMapEngine-
[RBPM] Unable to complete the CODE MAP refresh for entitlement:
cn=useraccount,cn=ad,cn=dset,ou=idm,o=system.
```

The first entry indicates that code map table refresh action is being initiated. The second entry specifies that it is unable to refresh the code map table for `cn=exchangemailbox,cn=ad,cn=dset,ou=idm,o=system` entitlement. The next line has the time interval when the entitlement will be refreshed. The next two lines specify that code map table was not refreshed for `cn=group,cn=ad,cn=dset,ou=idm,o=system` and `cn=useraccount,cn=ad,cn=dset,ou=idm,o=system` entitlements.

The code map refresh process can fail when either connected system or the Identity Vault is not up at the time of obtaining entitlement information. The trace logs the actual reason for failure.

When Multiple Users Try to Authenticate From Different Interfaces

OSP supports the OAuth2 specification and requires an LDAP authentication server. By default, Identity Manager uses Identity Vault (eDirectory) as an authentication server. When multiple users try to log in to OSP from different user interfaces of the identity applications, the users are redirected to the default landing page upon a successful login. When the access token expires within the login session, OSP validates the token and refreshes the session by generating a new access token without the user's involvement. Otherwise, it directs the user to the login page. Such a trace looks similar to the following:

```
2016-03-08 06:14:28,509 [http-bio-8443-exec-801] DEBUG
com.netiq.idm.auth.oauth.OAuthFilter- [RBPM] Original request going to RBPM is:
https://www.snet.acme.com:443/IDMProv/
workDashboard.do?apwaLeftNavItem=JSP_MENU_TASKS

2016-03-08 06:15:49,289 [http-bio-8443-exec-816] DEBUG
com.netiq.idm.auth.oauth.OAuthFilter- [RBPM] Original request going to RBPM is:
https://www.snet.acme.com:443/IDMProv/
2016-03-08 06:15:50,334 [http-bio-8443-exec-815] INFO
com.netiq.idm.auth.oauth.OAuthFilter- [RBPM] SSO Header issued by SSO Filter oauth
for User cn=Rob.Andrews,ou=Active,ou=People,o=acme.

2016-03-08 06:15:50,354 [http-bio-8443-exec-815] INFO
com.novell.common.auth.saml.AuthTokenGenerator- [RBPM] SAML Token is issued by the
request from SSO filter oauth

2016-03-08 06:15:50,414 [http-bio-8443-exec-815] INFO
com.novell.pwdmgt.util.PasswordHelper- [RBPM] [Login_Success]
cn=David.Scully,ou=Active,ou=People,o=acme successfully logged in.

2016-03-08 06:17:53,520 [http-bio-8443-exec-819] DEBUG
com.netiq.idm.auth.oauth.OAuthFilter- [RBPM] Original request going to RBPM is:
https://www.snet.acme.com:443/IDMProv/

2016-03-08 06:17:55,194 [http-bio-8443-exec-811] INFO
com.netiq.idm.auth.oauth.OAuthFilter- [RBPM] SSO Header issued by SSO Filter oauth
for User cn=neil.smith,ou=Active,ou=People,o=acme.
2016-03-08 06:17:55,204 [http-bio-8443-exec-811] INFO
com.novell.common.auth.saml.AuthTokenGenerator- [RBPM] SAML Token is issued by the
request from SSO filter oauth

2016-03-08 06:17:55,234 [http-bio-8443-exec-811] INFO
com.novell.pwdmgt.util.PasswordHelper- [RBPM] [Login_Success]
cn=neil.smith,ou=Active,ou=People,o=acme successfully logged in.

2016-03-08 06:20:56,616 [http-bio-8443-exec-813] DEBUG
com.netiq.idm.auth.oauth.OAuthFilter- [RBPM] Original request going to RBPM is:
https://www.snet.acme.com:443/IDMProv/

2016-03-08 06:21:02,129 [http-bio-8443-exec-823] INFO
com.netiq.idm.auth.oauth.OAuthFilter- [RBPM] SSO Header issued by SSO Filter oauth
for User cn=dean.gardner,ou=Active,ou=People,o=acme.

2016-03-08 06:21:02,149 [http-bio-8443-exec-823] INFO
com.novell.common.auth.saml.AuthTokenGenerator- [RBPM] SAML Token is issued by the
```

```
request from SSO filter oauth

2016-03-08 06:21:02,216 [http-bio-8443-exec-823] INFO
com.novell.pwdmgt.util.PasswordHelper- [RBPM] [Login_Success]
cn=dean.gardner,ou=Active,ou=People,o=acme successfully logged in.

2016-03-08 06:24:28,626 [http-bio-8443-exec-830] DEBUG
com.netiq.idm.auth.oauth.OAuthFilter- [RBPM] Original request going to RBPM is:
https://www.snet.acme.com:443/IDMProv/
workDashboard.do?apwaLeftNavItem=JSP_MENU_TASKS

2016-03-08 06:24:40,547 [http-bio-8443-exec-814] WARN
com.netiq.idm.auth.oauth.OAuthManager- [RBPM] Token validation failed. HTTP status
code: 401 Detail message from authentication server: The access token is expired.
```

This trace indicates that the user is accessing the application after some idle time. The last message indicates that the token has expired. When the user tried to log in again, the token failed the validation and as a result the user cannot be logged in.

When an E-Mail Approval Notification is Not Delivered

Sometimes an e-mail notification is not delivered due to errors in the connection between the client and the mail server. When this occurs, the trace looks similar to the following:

```
2016-03-08 07:42:41,575 [NOTIFICATION THREAD] ERROR
com.novell.soa.notification.impl.NotificationThread- [RBPM] Error sending email.
com.novell.soa.notification.impl.NotificationException: Error sending email.
  at com.novell.soa.notification.impl.MailEngine.send(MailEngine.java:347)
at
com.novell.soa.notification.impl.NotificationThread.run(NotificationThread.java:96
)
Caused by: javax.mail.MessagingException: 421 Too many errors on this connection--
-closing

  at com.sun.mail.smtp.SMTPTransport.issueCommand(SMTPTransport.java:879)
at com.sun.mail.smtp.SMTPTransport.mailFrom(SMTPTransport.java:599)
  at com.sun.mail.smtp.SMTPTransport.sendMessage(SMTPTransport.java:319)
at com.novell.soa.notification.impl.MailEngine.send(MailEngine.java:344)
  ... 1 more
```

When an e-mail approval notification is not delivered, the first step should be to look at the logs and determine whether the connection is proper, mail server is running and accessible. Sometimes the e-mail fails to comply with e-mail template and fails to deliver.

When a Role Is Requested

When a role is requested in the identity applications, Identity Manager creates the role object in the Identity Vault. The Role and Resource Service driver checks users for assigning this role, and then provisions the role to the assigned users. When this occurs, the trace prints messages similar to the following:

```
2016-03-08 08:43:10,660 [pool-2-thread-5] INFO
com.novell.idm.nrf.service.RoleManagerService- [RBPM] [Role_Request] Requested by
cn=David.Scully,ou=Active,ou=People,o=acme, Target DN:
CN=Kaitlin.Demore,OU=active,OU=People,O=acme, Source
DN:cn=PennDOT_Vehicle_Certification,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm
361,ou=services,o=acme, Request DN:cn=20160308084310-
15da49b28ddf4ee1b7d71b4ce220c080-
0,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm361,ou=services,
o=acme, Request Category: 10, Request Status: 0, Original Request Status: 0,
Correlation ID: UserApp#UserStartWorkflow#082080ec-5b10-498f-b87d-28825ab63278

2016-03-08 08:43:10,669 [pool-2-thread-5] INFO
com.novell.idm.nrf.service.RoleManagerService- [RBPM] [Role_Request] Requested by
cn=David.Scully,ou=Active,ou=People,o=acme, Target DN:
CN=Kaitlin.Demore,OU=active,OU=People,O=acme, Source DN:cn=PennDOT History and
Photos Users,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm
361,ou=services,o=acme, Request DN:cn=20160308084310-
3c5b20b79cc046bb8267a41cad88a96a-
1,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm361,ou=services,
o=acme, Request Category: 10, Request Status: 0, Original Request Status: 0,
Correlation ID: UserApp#UserStartWorkflow#082080ec-5b10-498f-b87d-28825ab63278

2016-03-08 08:43:10,678 [pool-2-thread-5] INFO
com.novell.idm.nrf.service.RoleManagerService- [RBPM] [Role_Request] Requested by
cn=David.Scully,ou=Active,ou=People,o=acme, Target DN:
CN=Kaitlin.Demore,OU=active,OU=People,O=acme, Source DN:cn=JWL Eligible
Agency,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm
361,ou=services,o=acme, Request DN:cn=20160308084310-
fc24ac874aca4fc8b1db0e1d7662d9b3-
2,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm361,ou=services,
o=acme, Request Category: 10, Request Status: 0, Original Request Status: 0,
Correlation ID: UserApp#UserStartWorkflow#082080ec-5b10-498f-b87d-28825ab63278

2016-03-08 08:43:10,712 [pool-2-thread-5] INFO
com.novell.idm.nrf.service.RoleManagerService- [RBPM] [Role_Request] Requested by
cn=David.Scully,ou=Active,ou=People,o=acme, Target DN:
CN=Kaitlin.Demore,OU=active,OU=People,O=acme, Source DN:cn=DPW Recipient Address
Eligible Agency,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm
361,ou=services,o=acme, Request DN:cn=20160308084310-
411abb1e8f6f488182c37c8629275245-
3,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm361,ou=services,
o=acme, Request Category: 10, Request Status: 0, Original Request Status: 0,
Correlation ID: UserApp#UserStartWorkflow#082080ec-5b10-498f-b87d-28825ab63278

2016-03-08 08:43:10,737 [pool-2-thread-5] INFO
com.novell.idm.nrf.service.RoleManagerService- [RBPM] [Role_Request] Requested by
cn=David.Scully,ou=Active,ou=People,o=acme, Target DN:
CN=Kaitlin.Demore,OU=active,OU=People,O=acme, Source DN:cn=cj-users,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm
361,ou=services,o=acme, Request DN:cn=20160308084310-
211a591e09b04fbbb195fb14d7f4df07-
4,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm361,ou=services,
o=acme, Request Category: 10, Request Status: 0, Original Request Status: 0,
Correlation ID: UserApp#UserStartWorkflow#082080ec-5b10-498f-b87d-28825ab63278

2016-03-08 08:43:10,790 [pool-2-thread-5] INFO
com.novell.idm.nrf.service.RoleManagerService- [RBPM] [Role_Request] Requested by
cn=David.Scully,ou=Active,ou=People,o=acme, Target DN:
CN=Kaitlin.Demore,OU=active,OU=People,O=acme, Source DN:cn=JTS Eligible
Agency,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm
361,ou=services,o=acme, Request DN:cn=20160308084310-
```

```
5339699630814a91ac44530a244a02ba-
5,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm361,ou=services,
o=acme, Request Category: 10, Request Status: 0, Original Request Status: 0,
Correlation ID: UserApp#UserStartWorkflow#082080ec-5b10-498f-b87d-28825ab63278

2016-03-08 08:43:10,799 [pool-2-thread-5] INFO
com.novell.idm.nrf.service.RoleManagerService- [RBPM] [Role_Request] Requested by
cn=David.Scully,ou=Active,ou=People,o=acme, Target DN:
CN=Kaitlin.Demore,OU=active,OU=People,O=acme, Source DN:cn=Sentencing Guidelines
Eligible Agency,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm
361,ou=services,o=acme, Request DN:cn=20160308084310-
df2398cf36a042f0ac2241e693efb93c-
6,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,cn=idm361,ou=services,
o=acme, Request Category: 10, Request Status: 0, Original Request Status: 0,
Correlation ID: UserApp#UserStartWorkflow#082080ec-5b10-498f-b87d-28825ab63278

2016-03-08 08:43:10,800 [pool-2-thread-5] INFO
com.novell.soa.af.impl.LogEvent- [RBPM] [Role_Request_Submitted] Initiated by
cn=David.Scully,ou=Active,ou=People,o=acme, Process ID:
95f25f5f31224efcab1611fe0fc2471f, Process Name:
cn=newuserinvitation,cn=RequestDefs,cn=AppConfig,cn=UserApplication,cn=idm361,ou=s
ervices,o=acme, Activity: Activity6, Recipient:
CN=Kaitlin.Demore,OU=active,OU=People,O=acme, Correlation
ID:UserApp#UserStartWorkflow#082080ec-5b10-498f-b87d-28825ab63278, Submitted
Request:<?xml version="1.0" encoding="UTF-8"?><wfRoleRequest>
<attr name="sod-override-request">
<value>true</value>
</attr>
<attr name="target">
<value>CN=Kaitlin.Demore,OU=active,OU=People,O=acme</value>
</attr>
<attr name="action">
<value>GRANT</value>
</attr>
<attr name="targetType">
<value>USER</value>
</attr>
<attr name="roles">
<value>cn=PennDOT_Vehicle_Certification,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=ser
vices,o=acme</value>
<value>cn=PennDOT History and Photos Users,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=ser
vices,o=acme</value>
<value>cn=JWL Eligible Agency,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=ser
vices,o=acme</value>
<value>cn=DPW Recipient Address Eligible Agency,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=ser
vices,o=acme</value>
<value>cn=cj-users,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=ser
vices,o=acme</value>
<value>cn=JTS Eligible Agency,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=ser
vices,o=acme</value>
<value>cn=Sentencing Guidelines Eligible Agency,cn=Application
Access,cn=Level10,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=ser
vices,o=acme</value>
</attr>
<attr name="correlationId">
<value>UserApp#UserStartWorkflow#082080ec-5b10-498f-b87d-28825ab63278</value>
</attr>
<attr name="nrfRequest">
<value>cn=20160308084310-15da49b28ddf4ee1b7d71b4ce220c080-
```

```
0,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=services,o=acme</
value>
<value>cn=20160308084310-3c5b20b79cc046bb8267a41cad88a96a-
1,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=services,o=acme</
value>
<value>cn=20160308084310-fc24ac874aca4fc8b1db0e1d7662d9b3-
2,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=services,o=acme</
value>
<value>cn=20160308084310-411abb1e8f6f488182c37c8629275245-
3,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=services,o=acme</
value>
<value>cn=20160308084310-211a591e09b04fbbb195fb14d7f4df07-
4,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=services,o=acme</
value>
<value>cn=20160308084310-5339699630814a91ac44530a244a02ba-
5,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=services,o=acme</
value>
<value>cn=20160308084310-df2398cf36a042f0ac2241e693efb93c-
6,cn=Requests,cn=RoleConfig,cn=AppConfig,cn=UserApplication,ou=services,o=acme</
value>
</attr>
</wfRoleRequest>

2016-03-08 08:43:10,880 [pool-2-thread-5] INFO com.novell.soa.af.impl.LogEvent-
[RBPM] [Workflow_Forwarded] Initiated by System, Process ID:
95f25f5f31224efcab1611fe0fc2471f, Process Name:
cn=newuserinvitation,cn=RequestDefs,cn=AppConfig,cn=UserApplication,ou=services,o=
acme:205, Activity: Activity6, Recipient:
cn=Nancy.Wilmer,ou=Active,ou=People,o=acme
```

When a Role Is Listed in Role Catalog

When you issue a request to display a role in Role Catalog, the identity applications obtain the role object from the cache and then display the role information in Role Catalog.

```
2017-09-22 09:28:26,495 [DEBUG] RuntimeAuthorizationManagerService [RBPM]
Authorized result for:
cn=roleAdmin,cn=System,cn=Level20,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=User
Application Driver,cn=driverset1,o=system:role:read=true

2017-09-22 09:28:26,495 [DEBUG] CacheUtil [RBPM] Role object was found in cache:
cache-key-nrf-config

2017-09-22 09:28:26,495 [DEBUG] CacheUtil [RBPM] Role object RETRIEVED from cache:
cache-key-nrf-config

2017-09-22 09:28:26,509 [DEBUG] RuntimeAuthDelegatorBase [RBPM] Time to calculate
sel service access rights: 0milliseconds.

2017-09-22 09:28:26,509 [DEBUG] RuntimeAuthorizationManagerService [RBPM]
Authorized result for:
cn=secAdmin,cn=System,cn=Level20,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=User
Application Driver,cn=driverset1,o=system:role:read=true

2017-09-22 09:28:26,510 [DEBUG] CacheUtil [RBPM] Role object was found in cache:
cache-key-nrf-config

2017-09-22 09:28:26,510 [DEBUG] CacheUtil [RBPM] Role object RETRIEVED from cache:
cache-key-nrf-config

2017-09-22 09:28:26,522 [DEBUG] RuntimeAuthDelegatorBase [RBPM] Time to calculate
sel service access rights: 0milliseconds.

2017-09-22 09:28:26,522 [DEBUG] RuntimeAuthorizationManagerService [RBPM]
```

```
Authorized result for:
cn=resourceManager,cn=System,cn=Level20,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=
User Application Driver,cn=driverset1,o=system:role:read=true

2017-09-22 09:28:26,522 [DEBUG] CacheUtil [RBPM] Role object was found in cache:
cache-key-nrf-config

2017-09-22 09:28:26,522 [DEBUG] CacheUtil [RBPM] Role object RETRIEVED from cache:
cache-key-nrf-config

2017-09-22 09:28:26,533 [DEBUG] RuntimeAuthDelegatorBase [RBPM] Time to calculate
sel service access rights: 0milliseconds.

2017-09-22 09:28:26,533 [DEBUG] RuntimeAuthorizationManagerService [RBPM]
Authorized result for:
cn=roleManager,cn=System,cn=Level20,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=User
Application Driver,cn=driverset1,o=system:role:read=true

2017-09-22 09:28:26,534 [DEBUG] CacheUtil [RBPM] Role object was found in cache:
cache-key-nrf-config

2017-09-22 09:28:26,534 [DEBUG] CacheUtil [RBPM] Role object RETRIEVED from cache:
cache-key-nrf-config

2017-09-22 09:28:26,545 [DEBUG] RuntimeAuthDelegatorBase [RBPM] Time to calculate
sel service access rights: 0milliseconds.

2017-09-22 09:28:26,545 [DEBUG] RuntimeAuthorizationManagerService [RBPM]
Authorized result for:
cn=resourceAdmin,cn=System,cn=Level20,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=Us
er Application Driver,cn=driverset1,o=system:role:read=true

2017-09-22 09:28:26,545 [DEBUG] CacheUtil [RBPM] Role object was found in cache:
cache-key-nrf-config

2017-09-22 09:28:26,545 [DEBUG] CacheUtil [RBPM] Role object RETRIEVED from cache:
cache-key-nrf-config

2017-09-22 09:28:26,557 [DEBUG] RuntimeAuthDelegatorBase [RBPM] Time to calculate
sel service access rights: 0milliseconds.

2017-09-22 09:28:26,557 [DEBUG] RuntimeAuthorizationManagerService [RBPM]
Authorized result for:
cn=provAdmin,cn=System,cn=Level20,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=User
Application Driver,cn=driverset1,o=system:role:read=true

2017-09-22 09:28:26,557 [DEBUG] CacheUtil [RBPM] Role object was found in cache:
cache-key-nrf-config

2017-09-22 09:28:26,557 [DEBUG] CacheUtil [RBPM] Role object RETRIEVED from cache:
cache-key-nrf-config

2017-09-22 09:28:26,567 [DEBUG] RuntimeAuthDelegatorBase [RBPM] Time to calculate
sel service access rights: 0milliseconds.

2017-09-22 09:28:26,567 [DEBUG] RuntimeAuthorizationManagerService [RBPM]
Authorized result for:
cn=reportAdmin,cn=System,cn=Level20,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=User
Application Driver,cn=driverset1,o=system:role:read=true

2017-09-22 09:28:26,567 [DEBUG] CacheUtil [RBPM] Role object was found in cache:
cache-key-nrf-config

2017-09-22 09:28:26,567 [DEBUG] CacheUtil [RBPM] Role object RETRIEVED from cache:
cache-key-nrf-config
```

```
2017-09-22 09:28:26,582 [DEBUG] RuntimeAuthDelegatorBase [RBPM] Time to calculate
sel service access rights: 0milliseconds.

2017-09-22 09:28:26,582 [DEBUG] RuntimeAuthorizationManagerService [RBPM]
Authorized result for:
cn=provManager,cn=System,cn=Level20,cn=RoleDefs,cn=RoleConfig,cn=AppConfig,cn=User
Application Driver,cn=driverset1,o=system:role:read=true

2017-09-22 09:28:26,582 [DEBUG] JuiceHelper [RBPM] Kicked out of main loop with:
openR=-1 closeR=-1 i=0 idx=-1

2017-09-22 09:28:26,582 [DEBUG] JuiceHelper [RBPM] Setting RsMeta from cache:
METAlistRoles_defaultDescription&*~Name&*~enStatus_cn=uaadmin,ou=sa,o=data

2017-09-22 09:28:26,583 [DEBUG] VirtualDataModel [RBPM]
VDM.getEntityDefinition(String, Locale):sys-nrf-role

2017-09-22 09:28:26,583 [DEBUG] CacheUtil [RBPM] DAL Definition RETRIEVED from
cache: VDM_ENTITY_DEFINITION_sys-nrf-role

2017-09-22 09:28:26,583 [DEBUG] CacheUtil [RBPM] DAL Definition RETRIEVED from
cache: VDM_DEFINITION_ATTRIBUTE_LIST_sys-nrf-role

2017-09-22 09:28:26,583 [DEBUG] CacheUtil [RBPM] Role object was found in cache:
cache-key-nrf-config

2017-09-22 09:28:26,583 [DEBUG] CacheUtil [RBPM] Role object RETRIEVED from cache:
cache-key-nrf-config
```

The first log entry is the request to find the role object from the cache. The second log entry is the response that is returned, and it indicates that the role object was found. The object is then read and displayed in Role Catalog.

Schema Fails to Update When Updated Using a User Account That Was Not Used to Create the Schema

If Identity Applications use a different user account to update the schema than the account that was used to initially create it, the schema fails to update. It reports the following error in the `catalina.out` file:

```
2018-06-11 15:47:37,956 [localhost-startStop-1] INFO  liquibase- liquibase:
Clearing database change log checksums
2018-06-11 15:47:39,051 [localhost-startStop-1] ERROR
com.sssw.fw.servlet.EboBootServlet- [RBPM] Runtime exception initializing.
com.netiq.persist.PersistenceException: ORA-01031: insufficient privileges
                at
com.novell.soa.persist.DatabaseSchemaUpdate.unappliedChangeSets(DatabaseSc
hemaUpdate.java:365)
                at
com.novell.soa.persist.DatabaseSchemaUpdate.validateDatabaseSchema(Databas
eSchemaUpdate.java:229)
                at
com.sssw.fw.servlet.EboBootServlet.init(EboBootServlet.java:98)
                at
com.sssw.portal.servlet.EboPortalBootServlet.init(EboPortalBootServlet.jav
a:59)
                at
javax.servlet.GenericServlet.init(GenericServlet.java:158)
```

```
                at
org.apache.catalina.core.StandardWrapper.initServlet(StandardWrapper.java:
1284)
                at
org.apache.catalina.core.StandardWrapper.loadServlet(StandardWrapper.java:
1197)
                at
org.apache.catalina.core.StandardWrapper.load(StandardWrapper.java:1087)
                at
org.apache.catalina.core.StandardContext.loadOnStartup(StandardContext.jav
a:5229)
                at
org.apache.catalina.core.StandardContext.startInternal(StandardContext.jav
a:5516)
                at
org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:150)
                at
org.apache.catalina.core.ContainerBase.addChildInternal(ContainerBase.java
:901)
                at
org.apache.catalina.core.ContainerBase.addChild(ContainerBase.java:877)
                at
org.apache.catalina.core.StandardHost.addChild(StandardHost.java:649)
                at
org.apache.catalina.startup.HostConfig.deployWAR(HostConfig.java:1083)
                at
org.apache.catalina.startup.HostConfig$DeployWar.run(HostConfig.java:1880)
                at
java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
               at java.util.concurrent.FutureTask.run(FutureTask.java:266)
                at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:
1142)
                at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java
:617)
                at java.lang.Thread.run(Thread.java:745)
Caused by: liquibase.exception.LockException:
liquibase.exception.DatabaseException: Error executing SQL CREATE TABLE
ora_user_id.DATABASECHANGELOGLOCK (ID NUMBER(10) NOT NULL, LOCKED NUMBER(1)
NOT NULL, LOCKGRANTED TIMESTAMP, LOCKEDBY NVARCHAR2(255), CONSTRAINT
PK_DATABASECHANGELOGLOCK PRIMARY KEY (ID)): ORA-01031: insufficient
privileges
                at
liquibase.lockservice.StandardLockService.acquireLock(StandardLockService.
java:209)
                at
liquibase.lockservice.StandardLockService.waitForLock(StandardLockService.
java:148)
                at liquibase.Liquibase.clearCheckSums(Liquibase.java:886)
                at
com.novell.soa.persist.DatabaseSchemaUpdate.unappliedChangeSets(DatabaseSc
hemaUpdate.java:340)
                ... 20 more
```

To resolve this issue, set the following Java system properties for Liquibase in the `setenv` startup script at `/opt/netiq/idm/apps/tomcat/bin/ or c:\NetIQ\idm\apps\tomcat\bin:`

- -Dliquibase.schemaName={`schema_owner_id`}
- -Dliquibase.catalogName={`schema_owner_id`}

Checking the Status of Database Schema Validation

The identity applications database schema is validated when the administrator starts the identity applications server. The trace prints messages similar to the following:

```
2017-09-22 09:39:41,363 [INFO] DatabaseSchemaUpdate [RBPM] Connecting to PostgreSQL
version 9.4.10.

2017-09-22 09:39:41,375 [INFO] DatabaseSchemaUpdate [RBPM] Checking for database
schema

2017-09-22 09:39:41,401 [INFO] DatabaseSchemaUpdate [RBPM] Checking schema for
table af_resource_request_status....found

2017-09-22 09:39:41,422 [INFO] DatabaseSchemaUpdate [RBPM] Checking schema for
table af_role_request_status....found

2017-09-22 09:39:41,481 [INFO] DatabaseSchemaUpdate [RBPM] Checking schema for
table afactivity....found

2017-09-22 09:39:41,501 [INFO] DatabaseSchemaUpdate [RBPM] Checking schema for
table afactivitytimertasks....found

2017-09-22 09:39:41,576 [INFO] DatabaseSchemaUpdate [RBPM] Checking schema for
table afbranch....found

2017-09-22 09:39:41,612 [INFO] DatabaseSchemaUpdate [RBPM] Checking schema for
table afcomment....found

2017-09-22 09:39:41,651 [INFO] DatabaseSchemaUpdate [RBPM] Checking schema for
table afdocument....missing
```

The first log entry indicates that a connection to the database is initiated. The second log entry specifies that the database schema is being validated. The subsequent entries specify that the presence of database tables is being checked. The last trace entry indicates that `afdocument` table is not found.

Determining if Liquibase Changeset Has Executed

The Liquibase framework validates the database schema against the `changelog.xml` file, which contains the database changes. Liquibase first verifies whether all the tables are present in the schema. If any of the changesets was not executed, it indicates incomplete schema update and also identifies the changesets. Any schema changes made in the User Application are updated in the database when the User Application server is started and `com.netiq.idm.create-db-on-startup` flag is set to `true` in the `ism-configuration` properties file located by default in the `/netiq/idm/apps/tomcat/conf` directory. The database compares the existing schema with target schema and then updates the database schema. If the flag is not set, it reports the following message:

```
One or more required tables are missing. Check log for messages indicating
tables that were not found.
```

When Liquibase validates the database schema, it generates log entries similar to the following:

```
2017-09-22 09:39:47,693 [INFO] liquibase liquibase: Reading from
public.databasechangelog

2017-09-22 09:39:49,133 [INFO] liquibase liquibase: Reading from
public.databasechangelog

2017-09-22 09:39:49,427 [INFO] liquibase liquibase: Reading from
public.databasechangelog

2017-09-22 09:39:49,551 [INFO] liquibase liquibase: Reading from
public.databasechangelog

2017-09-22 09:39:50,095 [INFO] liquibase liquibase: Reading from
public.databasechangelog

2017-09-22 09:39:50,403 [INFO] liquibase liquibase: Reading from
public.databasechangelog

2017-09-22 09:39:50,514 [ERROR] EboPortalBootServlet [RBPM] Unexpected Runtime
Exception initializing servlet
java.lang.RuntimeException: com.netiq.common.i18n.LocalizedRuntimeException:
Schema is invalid. One or more required tables are missing. Check log for messages
indicating tables that were not found.
        at com.sssw.fw.servlet.EboBootServlet.init(EboBootServlet.java:115)
        at
com.sssw.portal.servlet.EboPortalBootServlet.init(EboPortalBootServlet.java:62)
        at javax.servlet.GenericServlet.init(GenericServlet.java:158)
        at
org.apache.catalina.core.StandardWrapper.initServlet(StandardWrapper.java:1183)
        at
org.apache.catalina.core.StandardWrapper.loadServlet(StandardWrapper.java:1099)
        at org.apache.catalina.core.StandardWrapper.load(StandardWrapper.java:989)
        at
org.apache.catalina.core.StandardContext.loadOnStartup(StandardContext.java:4913)
        at
org.apache.catalina.core.StandardContext.startInternal(StandardContext.java:5223)
        at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:150)
        at
org.apache.catalina.core.ContainerBase.addChildInternal(ContainerBase.java:752)
        at org.apache.catalina.core.ContainerBase.addChild(ContainerBase.java:728)
        at org.apache.catalina.core.StandardHost.addChild(StandardHost.java:734)
        at org.apache.catalina.startup.HostConfig.deployWAR(HostConfig.java:952)
        at
org.apache.catalina.startup.HostConfig$DeployWar.run(HostConfig.java:1823)
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
        at java.util.concurrent.FutureTask.run(FutureTask.java:266)
        at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:745)
Caused by: com.netiq.common.i18n.LocalizedRuntimeException: Schema is invalid. One
or more required tables are missing. Check log for messages indicating tables that
were not found.
        at
com.novell.soa.persist.DatabaseSchemaUpdate.validateDatabaseSchema(DatabaseSchemaU
pdate.java:245)
        at com.sssw.fw.servlet.EboBootServlet.init(EboBootServlet.java:99) ... 18
more

Sep 22, 2017 9:39:50 AM org.apache.catalina.core.StandardContext loadOnStartup
SEVERE: Servlet [PortalAggregator] in web application [/IDMProv] threw load()
exception java.lang.NullPointerException at
com.sssw.portal.manager.EboPortalManager.<init>(EboPortalManager.java:179)
```

You must restart the application server to apply the changes. You cannot bring up the identity applications until the schema validation succeeds.

When Assigning a Resource to a User That Does Not Exist

If a user no longer exists in the system, and a request is issued to assign a resource to that user, the trace records messages similar to the following:

```
2017-09-22 11:50:53,605 [DEBUG] DataItemEvaluator [RBPM] result: Add Resource To
User - Laptop

2017-09-22 11:50:53,607 [ERROR] VirtualDataAccess [RBPM] Error occurred checking
the object type for: cn=rocio,ou=users,o=data
javax.naming.NameNotFoundException: [LDAP: error code 32 - NDS error: no such entry
(-601)]; remaining name 'cn=rocio,ou=users,o=data'
        at com.sun.jndi.ldap.LdapCtx.mapErrorCode(LdapCtx.java:3161)
        at com.sun.jndi.ldap.LdapCtx.processReturnCode(LdapCtx.java:3082)
        at com.sun.jndi.ldap.LdapCtx.processReturnCode(LdapCtx.java:2888)
        at com.sun.jndi.ldap.LdapCtx.c_getAttributes(LdapCtx.java:1329)
        at
com.sun.jndi.toolkit.ctx.ComponentDirContext.p_getAttributes(ComponentDirContext.j
ava:235)
        at
com.sun.jndi.toolkit.ctx.PartialCompositeDirContext.getAttributes(PartialComposite
DirContext.java:141)
        at
com.sun.jndi.toolkit.ctx.PartialCompositeDirContext.getAttributes(PartialComposite
DirContext.java:129)
        at sun.reflect.GeneratedMethodAccessor432.invoke(Unknown Source)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at
com.sssw.fw.directory.realm.impl.jndildap.EboLdapContextProxyHandler.invokeMethod(
EboLdapContextProxyHandler.java:145)
        at
com.sssw.fw.directory.realm.impl.jndildap.EboLdapContextProxyHandler.invoke(EboLda
pContextProxyHandler.java:86)
        at com.sun.proxy.$Proxy155.getAttributes(Unknown Source)
        at
com.novell.srvprv.impl.vdata.model.VirtualDataAccess.getObjectType(VirtualDataAcce
ss.java:3943)
        at
com.novell.srvprv.impl.vdata.model.VirtualDataAccess.getObjectType(VirtualDataAcce
ss.java:4003)
        at
com.novell.srvprv.impl.vdata.model.VirtualDataModel.getObjectType(VirtualDataModel
.java:1405)
        at com.novell.soa.util.LdapUtil.isTypeOf(LdapUtil.java:109)
        at com.novell.soa.util.LdapUtil.isUser(LdapUtil.java:89)
```

The trace displays object unavailable errors only when errors occur while retrieving an object.

When Checking the Workflow Engine Heartbeat

The following are example entries that are logged when a user issues a request to check the state of the workflow engine.

```
2017-09-22 11:53:33,646 [TRACE] EngineStateDAO [RBPM] Updating heartbeat of Engine
Engine State: engineId = ENGINE, heartBeat=
2017-09-22 11:52:33.637, startTime= 2017-09-22 09:55:31.532, shutdownTime= 2017-
09-22 09:52:58.773, engineState= Running

2017-09-22 11:53:33,647 [DEBUG] HibernateUtil [RBPM] Beginning new transaction for
caller com.novell.soa.af.impl.persist.EngineStateDAO:141

2017-09-22 11:53:33,652 [TRACE] EngineStateDAO [RBPM] Engine heartbeat updated
successfully: Engine State: engineId = ENGINE, heartBeat= 2017-09-22 11:53:33.651,
startTime= 2017-09-22 09:55:31.532, shutdownTime= 2017-09-22 09:52:58.773,
engineState= Running

2017-09-22 11:53:33,660 [DEBUG] HibernateUtil [RBPM] Committed transaction for
caller com.novell.soa.af.impl.persist.EngineStateDAO:162

2017-09-22 11:53:33,660 [DEBUG] EngineImpl [RBPM] Heartbeat updated for engine:
ENGINE, time: 2017-09-22 11:53:33.651

2017-09-22 11:54:03,017 [DEBUG] HibernateUtil [RBPM] Beginning new transaction for
caller com.novell.soa.af.impl.timers.PendingActivityTimerTask:94

2017-09-22 11:54:03,018 [DEBUG] HibernateUtil [RBPM] Committed transaction for
caller com.novell.soa.af.impl.timers.PendingActivityTimerTask:94
```

If the workflow engine is not properly shutdown due to some reason, the identity applications assume that the engine is still running when you start it the next time. The following example traces are logged to indicate this situation.

```
2017-09-22 12:01:43,384 [WARN] EngineImpl [RBPM] Duplicate engine id detected. This
engine may not have been shutdown cleanly or another engine is running with engine-
id: ENGINE. Waiting 60000 ms for heartbeat to timeout.

2017-09-22 12:01:43,480 [INFO] EngineImpl [RBPM] Workflow Engine setState:
[RUNNING]
```

catalina.out File Does Not Rotate the Log on Linux

On Linux, logrotate utility handles the log rotation of `catalina.out` file. The log rotation configuration is stored in the `netiq-tomcat` file in `/etc/logrotate.d` directory. If logrotate is not scheduled to run daily, the logs are not rotated. NetIQ recommends you to rotate the logs at `12:00 a.m` (midnight).

If SELinux is configured to run in Enforcing mode, logrotate might not work as expected.

Workaround: Run the following command:

```
semanage fcontext -a -t var_log_t '/opt/netiq/idm/apps/tomcat/logs(/.*)?'
restorecon -Frvv /opt/netiq/idm/apps/tomcat/logs
```

Troubleshooting E-Mail Based Approval Issues

Empty E-Mail Based Approval Token in the Provisioning Request Mail

This can occur if E-Mail Based Approval is not enabled. For example, the feature is accidentally disabled while using the new e-mail templates in PRDs.

Check the configuration in the Identity Manager Dashboard and enable the feature.

User Application is Not Acting on E-Mails

Check whether the incoming mailbox is connected and reachable from the server where it is deployed. For more information, refer to the catalina.out logs.

Approve or Deny Link in E-Mail is Not Working

This can occur in the following cases:

- The e-mail client is not configured.
- Default application is not selected to send e-mails.

Approve/Deny links Missing from E-Mail after configuring E-Mail Based Approval

This occurs if the e-mail templates are not properly configured on the workflows.

Verifying if E-Mail Based Approval Starts Properly

If you are enabling the feature from the new Dashboard, a success message appears indicating that the feature has started properly without errors. Messages similar to the following are logged in the `catalina.out` file:

```
INFO com.novell.soa.notification.impl.EmailReceiverEngine- [RBPM]
Successfully started persistent JMS notification system for email based
approval EmailReceiver Notification Thread]
```

```
INFO com.novell.soa.notification.impl.EmailReceiverThread- [RBPM] Starting
asynchronous notification system
```

```
INFO com.novell.soa.notification.impl.EmailReceiverEngine- [RBPM] Mailbox
service for incoming mail started successfully without any warning
```

```
INFO com.novell.soa.notification.impl.EmailReceiverEngine- [RBPM] Email
based approval token cleanup service started successfully.
```

In case the feature is accidentally turned off, check the configuration in the Identity Manager Dashboard. You can also refer to the log for detailed information for each component of this feature such as JMS, incoming mail box connection, and cleanup service.

When is Server Restart Needed

On a cluster setup, if you made changes to the incoming mailbox properties or turned off E-Mail Based Approval, you may require to restart the cluster nodes other than the active node.

If you are continuously getting errors while trying to connect to the mailbox and the issue persists for hours, verify the connectivity between the mailbox and the host.

E-Mail Based Approval Token is Empty in the Provisioning Request E-Mail

E-Mail Based Approval is accidentally disabled while using the new e-mail templates in PRDs.

Troubleshooting Self Service Password Reset Issues

No Redirection to Challenge-Response Page When SSPR is Installed in a Distributed Environment That Supports http and https Communication

Issue: If identity applications or OSP use HTTPS communication and SSPR running on separate computer uses a non-SSL (HTTP) communication, SSPR does not display the challenge-response page. This occurs because the browser blocks the mixed-content access.

Workaround: Enable the mixed-content access option in your browser.

Unable to Unlock Account through SSPR

Issue: This issue occurs if NMAS or eDirectory Challenge Responses are stored only in eDirectory and not in SSPR.

Workaround: Force the users to setup the challenge questions in SSPR as follows:

1. Go to SSPR **Configuration Editor**.
2. Click **Modules > Enabled > Authenticated > Setup Security Questions > Force Response**.

SSPR Reports Error 5027 When Attempting to Access Configuration Manager through Internet Explorer

Issue: In a new installation or upgraded setup of Self Service Password Reset 4.x, when you successfully log in to the SSPR portal, accessing the Configuration Manager or Configuration Editor displays the following error:

```
2016-11-01T15:54:00Z, ERROR, http.PwmResponse, {21,uaadmin} 5027
ERROR_UNAUTHORIZED (Internet Explorer version is not supported for this
function. Please use Internet Explorer 11 or higher or another web
browser.) [151.155.213.181]
```

Workaround: Disable Compatibility View in Internet Explorer for the domain that hosts your SSPR web application.

SSPR Reports Out of Order Page Request Error

Issue: This issue occurs when you click the **Back** button from the SSPR page. SSPR displays an incorrect sequence message in the SSPR error log similar to the following:

```
ERROR, password.pwm.servlet.TopServlet, 5035
ERROR_INCORRECT_REQUEST_SEQUENCE (expectedPageID=3, submittedPageID=4,
url=<some sspr url>
```

Workaround: Disable the **Back** button detection from SSPR **Configuration Manager > Settings > Security > Web Security.**

NOTE: Changing this setting has no effect on end users.

Pressing Enter Button in SSPR's People Search Displays Locale Screen on Internet Explorer

Issue: If you press Enter in People Search on Internet Explorer 11 browser, the Locale screen appears.

This issue is not reported on other browsers such as Microsoft Edge, Mozilla Firefox, and Google Chrome.

Workaround: Perform one of the following actions:

- The search bar searches as you type. Therefore, you do not require to press Enter to search.
- Use a different browser.
- Close the locale prompt.

Troubleshooting Authentication Issues

OSP Login Request Example by Using REST Endpoints

The Identity Applications server supports APIs that expose all OAuth functionalities as endpoints for obtaining access tokens, and so forth.

The following is an example of the authentication sequence:

Browser requests Identity Applications Landing Page

```
GET    http://<ip address/DNS name of identity applications>:8180/idmdash/
```

Result: 200 text/html

The query includes a bunch of requests to obtain stylesheet (css), JavaScript, and so on.

Landing Page makes “who am I” REST call to the Identity Applications server

The Landing Page makes a request to Identity Applications with no authorization header as the landing page has no access token.

```
GET     http://prvdvnam850.namdom025.lab:8180/IDMProv/rest/access/users/
fullName
```

Authorization header: none

Result: 401 error

The Landing Page causes browser to go to OSP grant URL

As Identity Applications do not yet have an OAuth access token, it responds with a 401 error. This causes the Landing page to go to OSP to get an access token. Note that there are no OSP cookies yet.

```
GET
http://<ipaddress>:8180/osp/a/idm/auth/oauth2/
grant?response_type=token&redirect_uri=http:// <ipaddress>:8180/landing/
com.netiq.ualanding.index/
oauth.html&client_id=ualanding&state=spiffystate0.7645864660083901
```

Result: 200 text/html (The resulting page is the OSP login page)

The query includes a bunch of requests to obtain stylesheet (`css`) and favicon.

The result of the request to OSP (from the browser's point-of-view) is that a page is displayed with entry fields for the user's name and password. There are also cookies returned from OSP with the login page that will be sent by the browser in subsequent requests.

Browser POSTs user credentials from the login page

```
POST http://<ipaddress>:8180/osp/a/idm/auth/app/login?acAuthCardId=np-
contract-{%24default-card}&sid=1
```

Cookies: JSESSIONID 95...79 End Of Session

x-oidp-oauth2-1449687159117—1013136951 "Wtf...zx0~" End Of Session

x-oidp-session59303d34382c2d310 200-GX0...97kISI~ End Of Session

Result: 302 Redirect to OSP implicitcontinue

```
GET http://<ipaddress>:8180/osp/a/idm/auth/oauth2/
implicitcontinue?privateId=bb5b94976815f348307b&client_id=ualanding&irdpkg
=1449687159117--1013136951
```

Cookies: JSESSIONID 95...79 End Of Session

x-oidp-oauth2-1449687159117--1013136951 "Wtf...zx0~" End Of Session

x-oidp-session59303d34382c2d310 200-PP+...RzX0F6 End Of Session

Result: 302 Redirect to Identity Manager landing OAuth result page

After an internal redirect between the OSP pages, the result is a redirect to the `redirect_uri` parameter that was originally sent with the initial request to OSP.

Browser redirects to the Landing OAuth Result Page

```
GET    http://<ipaddress>:8180/idmdash/oauth.html
```

Cookies: x-oidp-session59303d34382c2d310 200-AZ...b/HQ~~ End Of Session

Result: 200

A fragment containing the access token (see section 4.2.2 of RFC 6749) is appended to the URL. The Landing page extracts the OAuth access token from this fragment. You cannot see this fragment because HTTP does not capture it.

Landing Page again makes the “who am I” request

The Landing Page again makes the “who am I” request, but this time with an authorization header as the Landing page has an access token.

```
GET http://<ipaddress>:8180/IDMProv/rest/access/users/fullName
```

Authorization header: Authorization bearer eHw...343

Result: 200 {"dn":"cn=mary,ou=users,o=data","name":"Mary Contrary"}

Managing the Size of oidPInstancedata Attribute

OSP creates oidpInstanceData attribute (Case Ignore, Single Valued String) for a user when the user logs in to the identity applications for the first time through OSP. OSP modifies this attribute each time a user logs in and out of the identity applications.

- When a user is logged in, OSP adds a login entry to the attribute in base64 encoded and encrypted value format.
- When the user logs out, OSP removes or modifies the login entry. When the user logs in again, OSP updates the entry. When the user logs out, OSP removes that login entry from the attribute.

When the user closes the browser instead of logging out, OSP does not remove the login entry because closing the browser does not involve a logout action. If the user continues to log in without logging out, the size of the entry grows large. This prevents OSP from updating the attribute and causes login failure for the user.

Note that a logout operation can only remove the entry for the login it is mapped or matched to. For example, if a user logs in three times and does not log out for these logins, and if the user logs in and out one more time, OSP removes this login entry.

If a user is not required to log out from the identity applications, perform one of the following actions to manage the size of the oidPInstancedata attribute:

- Shorten the validity period of the login entry for the user. This allows OSP to automatically remove the login entry for the user. The validity period is controlled by **Refresh token lifetime (hours)** setting for OSP in the ConfigUpdate utility. The default value to store a login entry is 48 hours (2 days). After making the change in the ConfigUpdate utility, restart the Tomcat server where OSP is deployed.

- Periodically delete the oidpInstanceData attribute from the user by using an LDAP based tool (iManager, jXplore, Apache Studio, and so on).
- Shorten the timeout value of the refresh token issued as part of OAuth "Resource Owner Password Grant". The `com.netiq.idm.osp.oauth.public.refreshTokenTTL` property in `ismconfiguration.properties` file specifies the timeout of this refresh token. After setting the value, restart the Tomcat server where OSP is deployed. The timeout is set to 2,592,000 seconds or 30 days by default, after which OSP automatically removes the login entry.

OSP Fails to Update the oidpInstanceData Attribute

OSP cannot update the oidpInstanceData attribute for a user if one of the following conditions is true:

- When the attribute is full with user's login entries.

  When the user logs in again, OSP fails to update the attribute with the new login entries because of insufficient space to store the entries. However, you can change the maximum length for storing the login entries based on your requirement.
- The user does not does not have sufficient rights in the Identity Vault.
- The OSP schema has not been extended in the Identity Vault and the user does not have this attribute.

Troubleshooting General Issues

You might encounter the following issues while working with the identity applications:

- "Mismatch of Certificates Used by Identity Manager Engine and User Application Causes Code (-9205) Error in vnd.nds.stream" on page 626
- "User Application Driver Fails to Communicate with the User Application Server on a Secured Connection" on page 626
- "Entitlement Configuration Error During Codemap Refresh" on page 627
- "Error After Logging Out of the Dashboard on Linux" on page 627
- "Bulk Import of Roles and Resources May Not Update the Permission Index" on page 627
- "Absence of Notification Templates Causes Workflow Error" on page 628
- "Error Occurs When You Add a New Application With a Logo" on page 628
- "User Application Driver Fails to Process Delete Events" on page 629
- "Card View Showing Only Default Attributes of the Recipient on Tasks Page" on page 629
- "Cannot Collect Identity Applications Audit Events Through CEF Auditing When SSL Is Enabled" on page 629
- "Unable to Search for Users While Requesting For Permissions on Behalf of Others" on page 630

Mismatch of Certificates Used by Identity Manager Engine and User Application Causes Code (-9205) Error in vnd.nds.stream

Issue: The Identity Manager drivers use Identity Manager engine's keystore instead of User Application's keystore to access the User Application. If these components use different certificates, drivers report an error message similar to the following when set at Trace level 5:

```
DirXML Log Event

Message: Code(-9205) Error in vnd.nds.stream://VAULT/TEST/DRIVERSET1/
DRIVER1/Publisher/POLICY#XmlData:133:
Couldn't request assignment of role: '<Role DN>' to identity: '<User DN>':
com.novell.nds.dirxml.soap.UserAppClientException:
java.lang.RuntimeException: javax.net.ssl.SSLHandshakeException:
sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to find
valid certification path to requested target
```

Workaround: Verify that the JRE used by the Identity Manager engine has the required certificate to connect to the User Application. Otherwise, import the certificate from the User Application.

1. Locate `cacerts` in the Identity Manager engine directory.

   For example, `/opt/novell/eDirectory/lib64/nds-modules/jre/lib/security/cacerts` on Linux.

2. Determine the certificate used by the User Application.

   2a. Navigate to the User Application keystore.

   For example, `/opt/netiq/idm/apps/jre/lib/security/cacerts`.

   2b. List the certificates by running the following command from the command line:

   ```
   keytool -list -v -keystore cacerts
   ```

3. (Conditional) If you have access to the certificate, import the certificate into Identity Manager engine's `cacerts` directory by running the following command:

   ```
   keytool -import -alias <newalias> -keystore cacerts -file
   certificate.der
   ```

4. (Conditional) If you do not have access to the certificate, export the certificate from the User Application's `cacerts` directory, and then import the certificate into Identity Manager engine's `cacerts` directory.

5. Restart the Identity Vault.

User Application Driver Fails to Communicate with the User Application Server on a Secured Connection

Issue: The User Application driver fails to communicate with the User Application server and returns a retry status error. This issue may occur if one of the following conditions is true:

- You are using Java 1.7.x in your environment.
- The User Application driver does not have the certificate required for the connection.

Workaround: Perform the following actions:

- Manually update your current Java version to version 1.8 Update 92 or later.
- Import the certificates from User Application into Identity Manager engine's JRE directory for use by the User Application driver. If your User Application server is protected by NetIQ Access Manager or a load balancer, add the certificates from Access Manager or the load balancer into Identity Manager engine's JRE directory.

Entitlement Configuration Error During Codemap Refresh

Issue: When a new resource is created in a driver, the resource is not added to the User Application after running the code map refresh for the driver. One of the reasons that can cause this issue is missing value of some of the parameters in the entitlement configuration of the driver. For example, `<entitlement data-collection="false" dn="CN=ExchangeMailbox,CN=AD Driver for Groups,CN=DriverSet,O=system" parameter-format="" resource-mapping="" role-mapping="">`.

User Application reports the following error in the `catalina.out` file:

```
2017-11-03 15:55:21,373 [http-bio-8443-exec-340] ERROR
com.novell.idm.nrf.persist.DirXMLDriverDAO- [RBPM] Error occurred parsing
the entitlement configuration XML: cn=EntitlementConfiguration,cn=AD
Driver for Groups,cn=DriverSet,o=system

java.lang.StringIndexOutOfBoundsException: String index out of range: 0
```

Workaround: Add the missing values in the entitlement configuration for the driver. For example, `<entitlement data-collection="false" dn="CN=ExchangeMailbox,CN=AD Driver for Groups,CN=DriverSet,O=system"parameter-format="idm4" resource-mapping="true" role-mapping="true">`.

Error After Logging Out of the Dashboard on Linux

Issue: On a Linux server, sometimes Identity Applications report the following error when you log out of the Dashboard.

```
5082 ERROR_STARTUP_ERROR (unable to write to applicationPath /opt/netiq/
idm/apps/sspr/sspr_data)
```

Workaround: Manually restart Tomcat.

Bulk Import of Roles and Resources May Not Update the Permission Index

Issue: Sometimes permission index is not updated if you are bulk importing roles or resources into the Identify Vault. This prevents the User Application's Role or Resource Catalogs to display the newly added roles or resources.

Workaround: Perform the following actions:

1. Stop the Tomcat application server where identity applications are deployed.
2. Delete the permission index from `/apps/tomcat/temp/permindex`.
3. Restart Tomcat.

Absence of Notification Templates Causes Workflow Error

Issue: Notification templates such as notification, email, and provisioning must reside in the Default Notification Collection folder in Identity Vault's Security container. If you perform any operations such as request permissions in the identity applications in absence of these templates, the following error is reported in the `catalina.out` file:

```
com.netiq.common.i18n.impl.LocalizedResourceResolverNoDefaultFoundExceptio
n: The resource resolver
com.novell.soa.notification.impl.vdx.LocalizedEmailTemplateResolver did
not return a resource for the default locale of en. It is required that a
resource exist for the default locale.
```

Workaround: Deploy the required packages for notification, email, and provisioning templates to the Identity Vault.

1. Open your project in Designer.
2. In the Outline pane, expand your project.
3. Right-click **Default Notification Collection**.
4. Select **Add All Templates**.
5. Select **Overwrite Existing Templates**, then click **OK**.
6. Right-click **Default Notification Collection**, select **Live**, and click **Deploy**.
7. Click **OK** to deploy.

Error Occurs When You Add a New Application With a Logo

Issue: When you click the **Add** button to add a new application with a logo (image), the following error appears:

```
Invalid image file uploaded
```

Workaround: Add the application without an image. Then, edit the newly added application to add an image as follows:

1. Ensure the user has write permissions for user home directory.

   For example: `/home/users/novlua/`
2. Log in to Identity Manager Dashboard and go to **Applications**.
3. Click Manage Applications icon.
4. Click **Edit** on the newly added application and add the logo (image).
5. Click **Save**.

User Application Driver Fails to Process Delete Events

If the User Application driver fails to establish a connection with the identity applications, the driver fails to process the delete operation and loops infinitely. You can confirm this by looking at the User Application driver startup and trace logs.

This issue typically occurs if the https certificates used by the identity applications are not available in the User Application driver's certificate store. The default certificate store for the driver is the Java `cacerts` directory (`/opt/novell/eDirectory/lib64/nds-modules/jre/lib/security/cacerts` or `<eDirectory install path>\jre\lib\security`).

Card View Showing Only Default Attributes of the Recipient on Tasks Page

Issue: In Identity Manager Dashboard, when you add new attributes to the **Card View** using **Settings** > **Customization** menu options, the changes are not applied on the Tasks page where Card View shows only default attributes of the **Recipient**.

Workaround: The **Card View** calls for and displays custom attributes listed under **Quick User Info** in **Directory Abstraction Layer** of **User Application Driver**. To display an attribute other than default, you must first add that attribute under **Quick User Info** Entities and deploy from Designer, then configure the **Customization Settings** in Identity Applications.

Cannot Collect Identity Applications Audit Events Through CEF Auditing When SSL Is Enabled

Issue: When you enable SSL connection between Identity Manager and a Security Event Log Management solution such as Sentinel, the CEF events for Identity Applications are not getting logged.

Workaround: Perform the following actions:

1. Log in to the server where you have installed Identity Applications.
2. Navigate to the `tomcat/conf` directory.
   - **Linux:** `/opt/netiq/idm/apps/tomcat/conf`
   - **Windows:** `C:\netiq\idm\apps\tomcat\conf`
3. Open the `idmuserapp_logging.xml` file.
4. Under the `com.netiq.idm.logging.syslog.CEFSyslogAppender` appender class, modify the `SyslogSslKeystoreFile` and `SyslogSslKeystorePassword` to the following:

   ```
   <param name="SyslogSslKeystoreFile"
   value="${com.netiq.idm.audit.cef.tlskeystore:/opt/netiq/idm/apps/jre/
   lib/security/cacerts}" />
   ```

   ```
   <param name="SyslogSslKeystorePassword"
   value="${com.netiq.idm.audit.cef.tlskeystore-
   password:KeystorePassword}" />
   ```
5. Save the changes.

6 Log in to Identity Manager Dashboard as an administrator.

7 Click **Configuration** > **Logging** > **Auditing Configuration** and select **Enable CEF Format**.

8 Specify the auditing server details.

9 To establish a secure communication with the auditing server, select **TCP** protocol and enable **Use TLS** option. Provide the **Keystore file path** and the **Keystore password**.

10 Click **Apply**.

11 Start the Tomcat service.

Unable to Search for Users While Requesting For Permissions on Behalf of Others

Issue: When requesting permissions for others, team managers and administrative users are unable to search for users on the New Request page. This occurs when the **User Search Lookup Attribute** or **User Search Default Attribute** includes custom (non-default) attributes on the Settings page.

Workaround: To resolve this issue, modify the trustee rights of individual users with team manager or administrative user roles in Identity Applications as described below:

1 Log in to iManager as an administrator.

2 Click the **View Objects** option.

3 In the **Tree** tab, click **data**.

4 Select the check box corresponding to the desired user name.

5 Go to **Actions > Modify Trustees**.

6 Click **Assigned Rights** option corresponding to the selected user name.

7 Click **Add Property > [All Attributes Rights] > OK**.

8 The user is assigned compare and read permissions by default. Assign additional rights as necessary.

9 Click **Done**.

10 Select **OK** or **Apply** to save the changes to the directory.

You can also change the trustee rights for all users under the `users.data` trustee name. Click **data > users > (current level)** check box in the **Tree** tab, then proceed to Step 5 through Step 10 in the procedure above.

Troubleshooting Multi-Threaded Role and Resource Service driver Issues

Use the following information to troubleshoot a multi-threaded Role and Resource Service driver:

- Information about the unique data set for which the request belongs to is appended in the driver command. It is also printed in the log. This information helps you determine if the driver policies correctly evaluated the disjoint set key for a request by comparing the key with the mapping table.

```
[02/22/18 12:34:14.586]:Role and Resource Service driver ST:Submitting
document to subscriber shim:
[02/22/18 12:34:14.586]:Role and Resource Service driver ST:
<nds dtdversion="4.0" ndsversion="8.x">
  <source>
    <product edition="Advanced" version="4.7.0.0">DirXML</product>
    <contact>NetIQ Corporation</contact>
  </source>
  <input>
    <nrf:resrequest Disjoint-Set="NETIQ"
dn="O=system\CN=driverset1\CN=User Application
Driver\CN=AppConfig\CN=RoleConfig\CN=ResourceRequests\CN=2018022212341
4-b4645d87e41043459ae4546fd000dcb8-0" event-id="idm-sles12-
195#20180222070414#1#1:bc9e66f7-ddf3-45bf-8b4d-f7669ebcf3dd"
xmlns:nrf="urn:dirxml:nrf"/>
  </input>
</nds>
```

- While storing an event in the driver storage, check the log for information about the disjoint set and the event IDs of all the commands for which processing is yet to complete.

```
[02/22/18 12:29:06.544]:Role and Resource Service driver :: Thread
ID:40 Processing request
DN: O=system\CN=driverset1\CN=User Application
Driver\CN=AppConfig\CN=RoleConfig\CN=Requests\CN=20180222122835-
9b90b601af744f5e890151b97d7fe7e4-0
[02/22/18 12:29:06.545]:Role and Resource Service driver ST:Receiving
DOM document from application.
[02/22/18 12:29:06.545]:Role and Resource Service driver ST:
<nds>
<source>
<product version="4.7.0.0">NetIQ Role and Resource Service Driver</
product>
<contact>NetIQ Corporation</contact>
</source>
<input>
<status level="success">Updating DirXML-DriverStorage attributes</
status>
<init-params event-id="storage">
<subscriber-state>
<nrf:request Disjoint-Set="NETIQ" dn="O=system\CN=driverset1\CN=User
Application
Driver\CN=AppConfig\CN=RoleConfig\CN=Requests\CN=20180222122835-
9b90b601af744f5e890151b97d7fe7e4-0" event-id="idm-sles12-
195#20180222065835#1#39:f1f354dc-81fa-4d8c-aa7c-dc54f3f1fa81"
xmlns:nrf="urn:dirxml:nrf"/>
<nrf:request Disjoint-Set="NETIQ" dn="O=system\CN=driverset1\CN=User
Application
Driver\CN=AppConfig\CN=RoleConfig\CN=Requests\CN=20180222122835-
de01c6eb63b34ec28ae66e1057e3f524-0" event-id="idm-sles12-
195#20180222065835#1#40:fdff5c1e-74d6-4415-9a61-1e5cfffdd674"
xmlns:nrf="urn:dirxml:nrf"/>
</subscriber-state>
```

- You can check the commands for which processing is yet to complete through iManager.
  1. In iManager, open the Identity Manager Administration page.
  2. Open the driver set that contains the multi-threaded Role and Resource Service driver.
  3. If the driver set is not listed on the **Driver Sets** tab, use the **Search In** field to search for and display the driver set.
  4. Click the driver set to open the Driver Set Overview page.
  5. Locate the driver icon, then click the upper right corner of the driver icon to display the **Actions** menu.
  6. Click **Edit Properties** to display the driver's properties page.
  7. Click **General > DirXML-DriverStorage**.
- Logs display the following information:
  - Disjoint key which the driver evaluates from the request command.
  - Information about the worker threads to which the driver will submit the request.
  - Thread Id's of the worker threads.

  Below is a sample log file content.

```
[02/23/18 11:59:34.977]:Role and Resource Service driver ST:: Thread
ID:65 Worker threads not found for the disjoint key. Disjoint Key: NETIQ
[02/23/18 11:59:34.979]:Role and Resource Service driver ST:: Thread
ID:65 Registered the worker threads with disjoint key. Disjoint Key:
NETIQ
<status event-id="idm-sles12-195#20180223062903#1#1:78daa132-558c-4d3f-
b088-32a1da788c55" level="success">Thread ID:65 Successfully updated
the event in the Driver Storage.
Request DN: O=data\OU=netiq\CN=netiq1</status>
```

- If the driver storage is full, information about event retry is printed in the logs. The log file contains entries similar to the following:

```
[01/20/18 18:53:45.484]:Role and Resource Service Driver ST:: Thread
ID:30 Cant update the command in Driver Storage. Driver storage is
full!!!
[01/20/18 18:53:45.484]:Role and Resource Service Driver ST:: Request
processing completed in Roles and Resource driver
[01/20/18 18:53:45.485]:Role and Resource Service Driver ST:Requesting
30 second retry delay.
[01/20/18 18:53:45.485]:Role and Resource Service Driver ST:
DirXML Log Event ------------------
     Driver:   \NOVELL_1\system\driverset1\Role and Resource Service
Driver
     Channel:  Subscriber
     Status:   Retry
     Message:  Code(-9006) The driver returned a "retry" status
indicating that the operation should be retried later. Detail from
driver: Thread ID:30 Cant update the command in Driver Storage. Driver
storage is full!!!
```

VII Appendix

The following sections provide additional reference information and advanced topics for the Identity Manager User Application.

- Appendix A, “Configuring the Identity Manager Approvals App,” on page 635
- Appendix B, “Schema Extensions for the User Application,” on page 647
- Appendix C, “JavaScript Search API,” on page 659
- Appendix D, “Trouble Shooting,” on page 669

A Configuring the Identity Manager Approvals App

The NetIQ Identity Manager Approvals app allows managers and resource owners to approve or deny requests remotely, using an iPhone or iPad with the iOS operating system or any device with Android operating system installed. Your users can see and work with the same approval tasks in the app that they would normally see in the User Application interface. All changes are synchronized between the Approvals app and the User Application.

This appendix provides information about configuring your environment to allow users to use the new interfaces. These sections are intended to provide necessary information to administrators who want to enable and configure the Approvals app in their environment.

Most users should not need to refer to this document, but should instead be able to install, configure, and use the app without additional instructions. For information about installing or using the Approvals app, see "Using the Identity Manager Approvals App" in the *NetIQ Identity Manager - User's Guide to the Identity Applications*.

Product Requirements

The Approvals app has the following prerequisites:

- **On the Identity Applications server:**
  - Identity Manager 4.5 Advanced Edition or later
  - Identity Manager Roles Based Provisioning Module 4.5 or later
  - Designer for Identity Manager 4.5 or later with User Application driver and latest User Application Base package installed
  - Enable SSL using valid Certificate Authority (CA) issued certificate. For detailed information on configuring and enabling SSL in your Identity Manager environment, see "Using SSL for Secure Communication" on page 581.
- **On the device:** Apple iPhone or iPad with Apple iOS 5, iOS 6, or iOS 7 operating system.

Enabling Non-Administrators to Use the Approvals App

If you want users who are not provisioning administrators on Identity Applications server to use the Approvals app, you must open the SOAP endpoints used by the server and the Approvals app to non-provisioning administrator users.

NOTE: Opening SOAP endpoints to non-provisioning administrator users does not compromise security. Identity Manager continues to enforce all other existing security checks.

Complete the following steps to open the SOAP endpoints on the Identity Applications server:

1. Stop the server.
2. Back up the existing `ism-configuration.properties` file.

   NOTE: By default, the `ism-configuration.properties` file is located at `/opt/netiq/idm/apps/tomcat/conf`.

3. Open the `ism-configuration.properties` file and change the following configuration file properties to the specified values:

Property	Value
`WorkflowService/SOAP-End-Points-Accessible-By-ProvisioningAdminOnly`	`false`
`WorkflowService/soap/addComment`	`false`
`WorkflowService/soap/getComments`	`false`
`VirtualDataService/soap`	`false`

4. Save and close the file.
5. Start the server.

Setting Up the Approvals App

Before your users can use the Approvals app, you must first configure your Identity Manager Roles Based Provisioning Module environment.

After installing the app, users can configure the app manually or automatically. Because manually configuring the Approvals app can be difficult, we recommend that administrators simplify the configuration process by providing users the necessary information as part of the provisioning process.

You provide configuration information to your users through a configuration link you customize for your Identity Manager Roles Based Provisioning Module environment. The structure of the configuration link is as follows:

```
idmapproval://settings/
?userid=Username&passwordInKeychain=Password&host=HostName&port=PortNumber
&
rbpmContext=Context&userContainer=UserContainer&timeout=Timeout&vdxUserEnt
ity=UserEntity&
vdxNameFormatAttribute=NameFormat&vdxFirstNameAttribute=FirstNameAttr&vdxL
astNameAttribute=
LastNameAttr&vdxPhotoAttribute=UserPhotoAttr&vdxPhotoAttributeLdap=PhotoLD
APAttr&vdxPhoneAttribute=
WorkPhoneAttr&vdxMobileAttribute=MobilePhoneAttr&vdxEmailAttribute=EmailAt
tr&namingAttribute=
NamingAttr&provAdminGetTasksWorkaroundInPlace=ProvisioningAdmin
```

The link must include settings specific to your environment, so that users can easily connect to the Identity Applications server from the Approvals app. However, none of the settings are explicitly required for the link. If you leave any setting values empty, each user must configure those settings on their device.

For example, if you want to provide a standard configuration link for all users in your environment, you would leave the `userid` and `passwordInKeychain` values empty:

```
idmapproval://settings/
?userid=&passwordInKeychain=&host=123.112.20.109&port=8180&
rbpmContext=IDMProv&userContainer=ou=users,o=data&timeout=5&vdxUserEntity=
user&vdxNameFormatAttribute=
FirstName%20LastName&vdxFirstNameAttribute=FirstName&vdxLastNameAttribute=
LastName&
vdxPhotoAttribute=UserPhoto&vdxPhotoAttributeLdap=photo&vdxPhoneAttribute=
TelephoneNumber&
vdxMobileAttribute=mobile&vdxEmailAttribute=Email&namingAttribute=cn&
provAdminGetTasksWorkaroundInPlace=YES
```

For detailed information about the configuration settings, see “Understanding Approvals App Settings” on page 637.

You can provide a configuration link to your users in one of the following ways:

- Customize and deploy the default “Request Mobile Approval App” process request definition (PRD) to your User Application. A user can log into the User Application and request access to the app using the PRD, which sends an email notification with a personalized configuration link that includes information specific to that user.
- Embed your custom configuration link in an HTML page hosted on a Web server in your environment. All users in your environment can navigate to the HTML page in a browser on their device and then click the configuration link.
- Create a QR code from the configuration link and embed the QR code in an HTML page. Users can use a QR code reader on their device to scan the code.

Understanding Approvals App Settings

An Approvals app configuration link can include the following settings:

Configuration Setting Name	Login Setting Description
`userid`	Specifies the user name the user uses to access the Identity Applications server.
`passwordInKeychain`	Specifies the password the user uses to access the Identity Applications server.
`host`	Specifies the fully qualified domain name or IP address of the Identity Applications server.
`port`	Specifies the HTTPS port the app uses to connect to the server.
`rbpmContext`	Specifies the context used when installing the User Application WAR file. The default value is `IDMProv`.
`userContainer`	Specifies the full DN of the Identity Vault container that contains the user LDAP entry. **NOTE:** If you specify a user container to use, the Approvals app uses that container. If you do not specify a user container, the app attempts to detect the appropriate user container in the Identity Vault, searching all containers and subcontainers starting with the user container dn specified when running the User Application Configuration (configupdate) utility. If your Roles Based Provisioning Module environment includes a number of user containers, we recommend that you specify the container you want the app to use. You can configure a provisioning request definition (PRD) like the default "Request Mobile Approval App" definition to easily provision and configure your mobile end users. For more information about customizing the default PRD, see "Customizing and Using the Default Approvals App Provisioning Request Definition" on page 639.
`timeout`	Specifies the number of seconds the app waits when attempting to connect to the server before cancelling the connection. The default value is 5 seconds.
`vdxUserEntity`	Specifies the LDAP entity that represents a user in the Identity Vault. The default value is `user`.
`vdxNameFormatAttribute`	Specifies the DAL attribute representation the app uses to format a user's full name. The default value is `FirstName%20LastName`.
`vdxFirstNameAttribute`	Specifies the name of the DAL attribute that represents a user's first name. The default value is `FirstName`.

Configuration Setting Name	Login Setting Description
vdxLastNameAttribute	Specifies the name of the DAL attribute that represents a user's last name. The default value is LastName.
vdxPhotoAttribute	Specifies the name of the DAL attribute that contains a user's photo. The default value is UserPhoto. **NOTE:** If a user does not have a picture configured in the Identity Manager or has configured their Identity Manager settings to not display a picture, the app displays a generic image instead.
vdxPhotoAttributeLdap	Specifies the name of the LDAP attribute that contains the photo of the user. The default value is photo.
vdxPhoneAttribute	Specifies the name of the DAL attribute that represents a user's work phone number. The default value is TelephoneNumber.
vdxMobileAttribute	Specifies the name of the DAL attribute that represents a user's mobile phone number. The default value is mobile.
vdxEmailAttribute	Specifies the name of the DAL attribute that represents a user's email address. The default value is Email.
namingAttribute	Specifies the naming DAL attribute used in the Identity Vault to describe a name. The default value is cn.
provAdminGetTasksWorkaroundInPlace	Specifies whether the user is a Provisioning Administrator on the Identity Applications server. The default value is YES.

Customizing and Using the Default Approvals App Provisioning Request Definition

As an administrator, you can use Designer to customize a generic "Request Mobile Approval App" PRD that your users can use, through the User Application, to request access to the Approvals application.

When a user requests access, Identity Manager then verifies that the user has the permissions required to access the mobile interface and that the Identity Applications server supports the application. If the server is not configured correctly or does not have the correct patch installed, the PRD generates a task for the provisioning administrator that lets the administrator know what needs to be fixed in order to enable use of the Approvals app.

After Identity Manager verifies the user and environment meet all requirements, the PRD triggers an email notification to the user. The user should open this email on the iPhone or iPad where the user has already installed the Approvals app.

This email notification includes a special `idmapproval://settings` link that automatically provides the settings the user needs to access the Approvals app from their device. The user clicks the link from their device and can then access their tasks through the Approvals app.

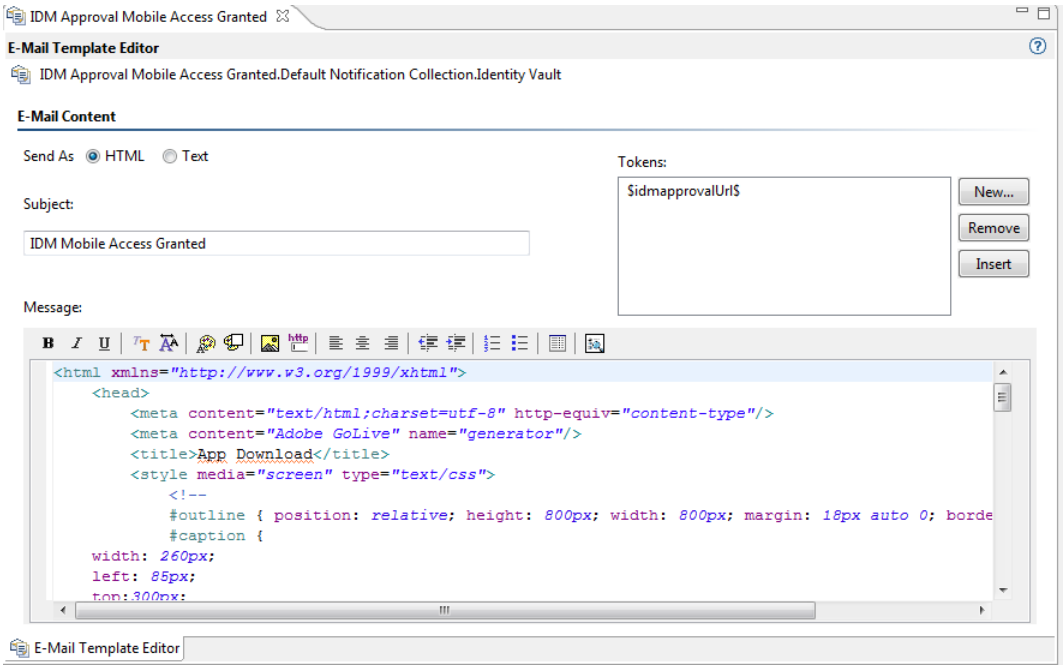
The default “Request Mobile Approval App” is included in the User Application Base package, which you can upgrade in Designer.

NOTE: The PRD and notification template provided in the User Application Base package are generic. Most administrators need to modify the generic PRD and template for their specific environments. However, we recommend that only users familiar with PRDs modify the default templates.

Customizing the Generic Notification Template

We recommend customizing the generic email notification template for your environment. To customize the default template to notify users they have access to the Approvals app and provide a link to automatically configure the app:

1. Ensure your Identity Manager environment meets all necessary requirements. For more information about prerequisites for using the Approvals app, see “Product Requirements” on page 635.
2. In Designer, ensure you have a valid User Application driver in production. If a User Application driver does not exist in your Designer installation, install the driver before proceeding.
3. Upgrade the User Application Base package to the latest available version and install any dependent packages. For information about upgrading packages in Designer, see Upgrading Installed Packages in the *NetIQ Designer for Identity Manager Administration Guide*.
4. In the Outline view, expand **Default Notification Collection**.
5. Right-click **IDM Approval Mobile Access Granted** and select **Edit**.
6. Modify the **Subject** field, if necessary.
7. In the **Message** field, modify the notification HTML as necessary for your environment. You can customize the email message text sent to your users, include graphics, or change the color and layout of the message to fit your company’s branding. The following image shows the default email message in the template editor:



WARNING: If you customize or modify the default notification template, do not remove or modify the token `$idmapprovalUrl$`, either in the Tokens list or in the HTML. The PRD uses the `$idmapprovalUrl$` token to provide the notification template a customized configuration link for the requesting user.

8 When finished making any customizations, close and save the notification template.

9 In the Outline view, right-click **IDM Approval Mobile Access Granted** and select **Live > Deploy**.

10 Click **Deploy**.

11 Click **OK**.

Customizing the Generic PRD

We recommend customizing the generic PRD for your environment. You can customize the category, workflow activities, entities, and forms. The PRD includes three forms by default:

- **request_form:** Request form users use to request access to the Approvals app.
- **approval_form:** Approval form managers use to approve or deny requests for access.
- **approval_form_prov_admin:** Approval form provisioning administrators use to fix issues with the provisioning server configuration.

To create and customize a PRD to automatically configure your users' settings in the Approvals app:

1 In the Outline view, navigate to the User Application driver.

2 Expand **User Application Driver > Provisioning Request Definitions > Accounts**.

3 Right-click **Request Mobile Approval Access** and select **Edit**.

4 Modify the **Display Name** and **Description** fields, if necessary.

5 (Optional) If you want to move the PRD from the default Accounts category, click the **Category** drop-down menu and select the category you want to use.

NOTE: Most users do not need to modify the **Status**, **Flow Strategy**, and **Process Type** fields for the generic PRD. We recommend only advanced users modify these fields.

6 (Optional) By default, the User DAL entity does not have an attribute configure for a user's mobile telephone number. If you do not currently have a `Mobile` attribute configured for User entities in your environment, you may need to add the attribute. Complete the following steps to add the attribute to the entity:

- **6a** In the Provisioning view, expand **User Application Driver > Directory Abstraction Layer > Entities**.
- **6b** Right-click **User** and select **Edit**.
- **6c** In the Data Abstraction Layer view, expand **Entities > User**.
- **6d** Right-click **User** and select **Add Attribute**.
- **6e** In the Add Attribute window, select the `mobile` attribute in the Available Attributes for Entity Class list.
- **6f** Click **Add Attribute** to move the attribute to the Entity Attributes list.
- **6g** Click **OK**.
- **6h** Close and save the Data Abstraction Layer.

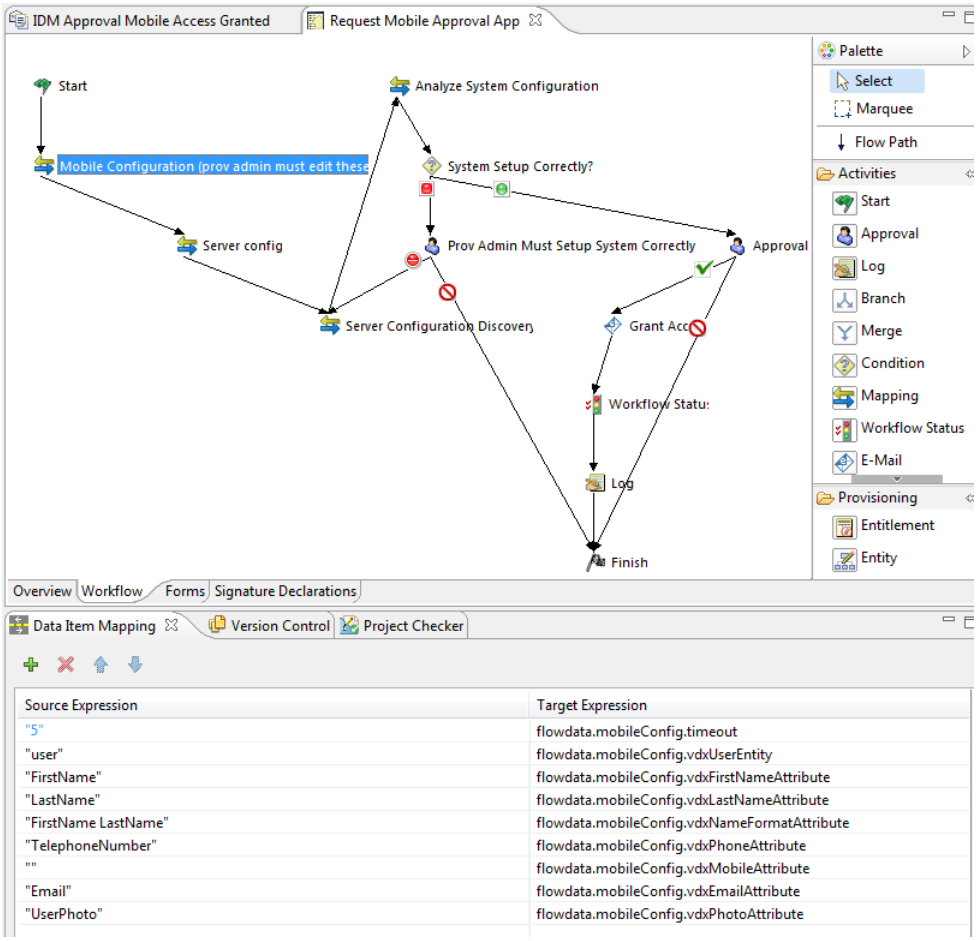
7 Click the Workflow tab.

8 Click **Mobile Configuration (prov admin must edit these)**.

9 Click **Data Item Mapping**.

10 Edit the data item mapping expressions for the Mobile Configuration workflow activity. Ensure that the data item mapping matches the way your DAL User entity is configured.

The following image shows the workflow activity and data item mapping:



11 (Optional) If you want to modify the default **Trustee rights** for the PRD, complete the following steps:

11a Click the Overview tab.

11b Click the plus icon.

11c Select the group or user you want to be able to request access to the Approvals app.

NOTE: By default, the PRD trustee rights are set to [ROOT]. This default setting allows all users to request access to the Approvals app. Administrators can configure the trustee rights to limit access to only certain users, if necessary.

11d Click **OK**.

12 (Optional) If you want to customize the default PRD request and approval forms, complete the following steps:

12a In the Forms view, click the name of the form.

12b Modify the fields in the Form Controls window, as necessary.

12c Click the Preview icon to view the form.

12d Click **OK** when finished.

13 When finished, close and save the Request Mobile Approval App tab.

14 In the Outline view, right-click **Request Mobile Approval App** and select **Sync to Package**.

15 Right-click **Request Mobile Approval App** and select **Live > Deploy**.

16 Click **Deploy**.

17 Click **OK**.

Creating and Deploying a Custom Configuration Link

If you want to provide a “generic” set of configuration settings to any user who installs the Approvals app, you can embed a configuration link in an HTML page on a Web server your users can access.

Include the standard configuration link syntax in a link, as in the following example:

```
<a href="idmapproval://settings/
?userid=&passwordInKeychain=&host=123.112.20.109&port=8180&rbpmContext=IDM
Prov&
userContainer=ou=users,o=data&timeout=5&vdxUserEntity=user&vdxNameFormatAt
tribute=FirstName%20LastName&
vdxFirstNameAttribute=FirstName&vdxLastNameAttribute=LastName&vdxPhotoAttr
ibute=UserPhoto&vdxPhotoAttributeLdap=
photo&vdxPhoneAttribute=TelephoneNumber&vdxMobileAttribute=mobile&vdxEmail
Attribute=Email&namingAttribute=cn&
provAdminGetTasksWorkaroundInPlace=YES">Configure Approvals App</a>
```

Unless you create a custom link for one specific user, most configuration links should leave the `userid` and `passwordInKeychain` values blank, providing the Identity Applications server information and Identity Vault information users need to be able to use the app.

A user clicks the link, and the link automatically configures the app with any settings you include in the link. The user then manually configures their Username and Password settings within the app.

Creating and Deploying a Custom Configuration QR Code

If your users cannot access their work email from their devices, you can create a QR code from the Approvals app configuration link and email that code to your users.

You can use any QR code generator you want to create the code, generating the code using a configuration link customized for your environment. Embed the code in an HTML page on a Web server your users can access.

For the example provided in “Setting Up the Approvals App” on page 636, the QR code could look like the following image:



A user can then install the app, open the email on their work computer, and use a QR code reader on their device to scan the code displayed on the screen.

The QR code acts as a configuration link, automatically configuring the app with any settings you include in the link. In most environments, your users need to then manually configure their Username and Password settings within the app.

Optimizing Designer Forms for the Approvals App

The Approvals app renders Designer forms using either native iOS controls or HTML, depending on the complexity of each specific form. Native iOS controls provide a more standard look and feel to forms, while HTML-rendered forms look similar to forms in the User Application interface.

When creating new forms in Designer, we recommend simplifying forms as much as possible so that the app uses native iOS controls.

You can also configure your forms to display a more complex version of the form in the User Application and a less complex version in the Approvals app, using the suffix `_mobile`.

For example, if you have an Approval activity form called `approveLaptop`, you can create a new form called `approveLaptop_mobile` that acts as a simplified version of the original Approval activity form. In order for data item mapping to function correctly, the `_mobile` form must include the same fields as the original. We recommend you keep both versions of the form synchronized.

The following steps can help you optimize your forms so the app can render using iOS controls:

1. Ensure the Identity Applications server has the correct version and patch installed. The server must have version 4.0.2 Patch B or later installed.
2. Ensure the form has no scripts.
3. Ensure the form contains only fields with the following supported data types and control types:
   - `boolean`: any control type
   - `date`: any control type
   - `time`: any control type
   - `decimal` or `integer`: `Text` control type only
   - dn: `DNDisplay` or read-only `MVEditor` control types only
   - `string`: `Text`, `Password`, `Title`, `TextArea`, or read-only MVEditor control types only

For more detailed information about creating forms in Designer, see "Creating Forms for a Provisioning Request Definition," in the *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.

Understanding Language Support in the Approvals App

The Approvals app includes localized text for all built-in text strings. For example, the titles displayed at the top of a view within the app are available in multiple languages, depending on the user's locale. Approvals app strings are provided in the following languages, by default:

- Chinese (Simplified)
- Chinese (Traditional)

- Danish
- Dutch
- English
- French
- German
- Italian
- Japanese
- Portuguese (Brazilian)
- Russian
- Spanish
- Swedish

As an administrator, you can also localize the form text displayed in the Approvals app. For example, the Approvals app does not provide localized text for specific Approval tasks. You must localize text strings for each of your PRDs, including form text, using Designer. For information about localizing objects in Designer, see "Localizing Provisioning Objects," in the *NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications*.

B Schema Extensions for the User Application

This section describes the schema extensions used by the User Application.

Attribute Schema Extensions

Attribute Name	Description
srvprvAllowMgrInitiate	A flag that indicates if the manager is allowed to initiate a provisioning request.
srvprvAllowMgrRetract	A flag to indicate if the manager is allowed to retract a provisioning request.
srvprvAllowMgrSetAvailability	A flag that indicates whether the manager can set a proxy for the team.
srvprvAllowMgrSetDelegate	A flag to indicate if the manager is allowed to set delegates for a provisioning request.
srvprvAllowMgrSetProxy	A flag to indicate if the manager is allowed to set a team proxy.
srvprvAllowMgrTaskClaim	A flag to indicate if the manager is allowed to claim a provisioning approval task.
srvprvAllowMgrTaskReassign	A flag to indicate if the manager is allowed to reassign a provisioning approval task.
srvprvAllRequests	A flag to indicate if the assignment covers all provisioning request definitions for a team.
srvprvAOLIMAddress	AOL IM address.
srvprvAssetRef	Representation of the aggregate asset properties for a named asset associated to a user via the srvprvAssetRecipientAux class.
srvprvAssignExpiration	Time at which a proxy or delegate assignment expires.
srvprvAssignFromContainer	Container subjects of a proxy or delegate assignment.
srvprvAssignFromGroup	Group subjects of a proxy or delegate assignment.
srvprvAssignFromUser	User subjects of a proxy or delegate assignment.
srvprvAssignStartTime	Time at which a delegation assignment takes effect.
srvprvAssignToRelationship	A target relationship of a delegate assignment.

Attribute Name	Description
srvprvAssignToUser	The User targets of a proxy or delegate assignment.
srvprvAutoDisplayTeam	Automatically display team members.
srvprvCapabilities1-5	Listing of skills for a user.
srvprvCategoryKey	Associates a given Provisioning Request Definition to a set of provisioning categories. Values are keys to a srvprvChoice instance.
srvprvCurrentDelegatees	The delegations associated with a user.
srvprvCurrentDelegators	The delegations associated with a user.
srvprvDefault	The default .
srvprvDelegateeDef	The delegates definition DN.
srvprvDelegationDef	The delegation definition DN.
srvprvDelegators	The users who are defined as delegators by this assignment.
srvprvEntitlementRef	Reference to a DirXML-Entitlement.
srvprvEntityType	Specifies Directory Abstraction Layer Entity definition type.
srvprvFlowStrategy	Specifies the flow invocation strategy to be used for the Provisioning Request Definition.
srvprvGrant	Flag which if true specifies that the Provisioning Request Definition supports a Grant operation.
srvprvGroupwiseIMAddress	Groupwise IM address.
srvprvHideAttributes	Flag indicating if certain attributes should be hidden and not displayed.
srvprvHideUser	Flag indicating if the user should be hidden when search list queries are executed.
srvprvIMAddress	Instant Messenger address.
srvprvIsTaskManager	Indicates if user is a task group manager.
srvprvLocalizedDescrs	Provides set of localized description strings for the provisioning web applications, Designers and iManager.
srvprvLocalizedNames	Provides set of localized display name strings for the provisioning web applications, Designers and iManager.
srvprvManager	Indicates users who are managers.
srvprvManagerGroup	Indicates a group containing managers.
srvprvManagerNotMember	Indicates that the manager is not a member of the team.

Attribute Name	Description
srvprvMember	Indicates users who are team members.
srvprvMemberContainer	The name of the container containing team members.
srvprvMemberGroup	The name of the group containing team members.
srvprvMemberRelationship	The name of the directory abstraction layer relationship that determines members based attribute in manager object.
srvprvModified	Flag to indicate changes to definitions object instances in the directory model container.
srvprvNotificationPrefs	Defines the set of notification types users want to receive.
srvprvPreferredLocale	Users preferred locale.
srvprvProcessXML	XML document representing a Provisioning process definition including Workflow and Provisioning Action.
srvprvQueryList	List of saved query/search criteria.
srvprvRelationship	Defines relationships between objects in the identity vault.
srvprvRequest	Exposes one item to be granted or revoked, including the workflow process which defines the run-time aspects of the Workflow and Provisioning Target.
srvprvRequestDefName	The provisioning request definition name associated with a delegate definition.
srvprvRequestScope	The scope of provisioning requests.
srvprvRequestXML	XML document representing the initial request form and its data bindings.
srvprvRevoke	If true, this flag specifies that the Provisioning Request Definition supports a Revoke operation.
srvprvStatus	Specifies the status of the Provisioning Object Supported values.
srvprvTaskGroups	Groups for which the user is a task manager.
srvprvTaskManager	Task manager of the task group.
srvprvTaskScopeAddressee	The addressee’s task scope.
srvprvTaskScopeRecipient	The recipient’s task scope.
srvprvTeam	The container for team definitions.
srvprvUser	The users associated with a delegation assignment.
srvprvUUID	Unique identifier for portlet.

Attribute Name	Description
srvprvYahooIMAddress	Yahoo* IM address.

Objectclass Schema Extensions

Objectclass Name	Description
srvprvAppConfig	Container for application configuration objects of the Provisioning System to which its DirXML-Driver parent connects.
srvprvAppDefs	Container for configuration objects used to initialize the Provisioning run-time environment, such as s for the Identity Portal.
srvprvAssetRecipientAux	Records the provisioning of non-IT assets on a user.
srvprvChoice	Enumeration of values that can be assigned to a particular attribute, used in a query, for use in the Identity Portlets and other Web Application components.
srvprvChoiceDefs	Container for Directory Abstraction Layer Choice definitions, to be exposed by the Identity Portlets and Web Applications.
srvprvDelegateeAssignment	Delegates assignment definition.
srvprvDelegateeDefs	Container for delegates definitions.
srvprvDelegationAssignment	Delegation or availability assignment definition.
srvprvDelegationDefs	Container for delegation and delegators definitions.
srvprvDelegatorAssignment	Delegation or availability assignment definition.
srvprvDirectoryModel	Container for Directory Abstraction Layer meta-level objects, selected contents of the directory to be exposed by the Identity Portlets and Web Applications.
srvprvDirectoryModelConfig	Runtime Directory Abstraction Layer configuration parameters.
srvprvEntity	Defines a view of selected attributes for defined classes in the directory, used by the Identity Portlets and other Web Application components.
srvprvEntityAux	Standard ObjectClass.
srvprvEntityDefs	Container for Directory Abstraction Layer Entity definitions, to be exposed by the Identity Portlets and Web Applications.
srvprvProxyAssignment	Proxy assignment definition.

Objectclass Name	Description
srvprvProxyDefs	Container for proxy definitions.
srvprvQuery	Directory abstraction layer query definition.
srvprvQueryDefs	Container for directory abstraction layer query definition.
srvprvRelationship	Defines relationships between objects in the directory, for use in the Identity Portlets and other Web Application components.
srvprvRelationshipDefs	Container for Directory Abstraction Layer Relationship definitions, to be exposed by the Identity Portlets and Web Applications.
srvprvRequest	Exposes one item to be granted or revoked, including the workflow process which defines the run-time aspects of the Workflow and Provisioning Target.
srvprvRequestDefs	Container for Provisioning Request Definitions, the set of items to the Web Application run-time.
srvprvResource	Defines the set of directory assignments to execute for a provisioning fulfillment operation (either Grant or Revoke).
srvprvResourceDefs	Container for Provisioning Target definitions, including design-time descriptions plus any template or unused targets.
srvprvService	Describes how to invoke a specific Web Service from an Workflow This includes specification of input and return values.
srvprvServiceDefs	Container for Service Definition objects, which wrap Web Services called by Workflows.
srvprvTaskGroupAux	Service provisioning task group.
srvprvTeam	Team for provisioning request management.
srvprvTeamDefs	Container for team definitions.
srvprvTeamRequest	Team provisioning requests.
srvprv	Object.
srvprvUserAux	Service provisioning user entity.
srvprvWebAppConfig	Web Application configuration object.
srvprvWorkflow	Defines the network of activities including traversal conditions to be executed in order to obtain approval for a provisioning action.
srvprvWorkflowDefs	Container for Workflow objects, including design-time descriptions plus any template or unused flows.

Resource Definition Object (nrfResource)

The schema object that contains provisioning resource definitions.

Table B-1 *Resource Definition Object Schema Definition*

Attribute Name	Description
nrfLocalizedName	The localized name of the resource.
nrfLocalizedDescrs	The localized description of the resource.
Owner	The owner of the resource. It is the DN of an inetOrgPerson user.
nrfRequestDefGrant	Provisioning request definition used for approving the granting of a resource assignment.
nrfRequestDefRevoke	Provisioning request definition usef for approving the revocation of a resource assignment.
nrfEntitlementRef	IDM entitlement associated with the resource. Supports embedding of dynamic parameter macros to allow users to specify values at request time.
nrfApprovers	Resource approvers. Order of approvers is maintained by an integer in the second element.
nrfQuorum	Used to support quorum approvals in tempated PRDs. This is the quorum condition. Can be percentage or number of approvers required.
nrfDynamicParameters	XML document that describes allowable parameter values that can be specified at request time when the resource is being granted.
nrfCategoryKey	Used to categorize resource.
nrfAllowAprOveride	Allow requesting system (such as role provisioning) to override approval of the resource provisioning.
nrfAllowMulti	Allow the resource to be assigned to the same user multiple times.

Resource Request Object (nrfResourceRequest)

The schema object whose instances contain a resource request object. The resource request object is used by the resource driver to provision the resource.

Table B-2 *Resource Request Object Schema Definition*

Attribute	Description
nrfRequestDate	Date-time resource request started.
nrfCategory	10-Resource To User Add
	15 - Resource to User Remove

Attribute	Description
nrfResource	DN of resource to grant or revoke.
nrfEntitlementRef	Entitlement reference value of the resource being granted. This value is copied from the resource definition with parameter values populated at the time of the request.
nrfTargetDN	DN of user who will be granted the resource or from whom the resource will be revoked.
nrfRequester	DN of user or role that requested assignment.
nrfStatus	Status of request. Valid codes are described in “Resource Request Status Codes (nrfStatus)” on page 653.
nrfDescription	Description/Comment of the resource request.
nrfRequestDef	Provisioning request definitoin used for approving the role
nrfApprovers	Resource approvers. Order of approvers can be maintained by an integer in the second element.
nrfQuorum	Used to support quorum approvals in templated PRDs. The quorum condition can be percentage or numbers of approvers required.
nrfApprovalInfo	Holds approval data needed by resource view and reports.
nrfApprovalProcessid	Workflow process instance ID for resource assignment approval.

Resource Request Status Codes (nrfStatus)

Table B-3 *Valid Resource Request (nrfStatus) Status Codes*

Status Code	Key	Description
01	New Request	Initial value when request is created
12	Approval_Retry	
13	Pending_Approval_RETRY	
15	Approval Pending	Set by driver after successful assignment/ revocation workflow.
20	Approved	Set by resource assignment/revocation workflow when approved.
30	Provision/Deprovision	Set by driver after all necessary approvals have been approved and role activation time has been reached.
50	Provisioned/Deprovisioned	Set by driver after role has been provisioned or deprovisioned.
70	Cancel	Request cancellation
75	Cancelled	Cancellation request completed.

Status Code	Key	Description
80	Provisioning Error	Set by driver when an error occurred during provisioning or deprovisioning.
95	DeniedSet	Set by assignment/revocation workflow when approved.
100	CleanupSet	When nrfResourceRequest workflow should be deleted.

Role Definition Object (nrfRole)

The schema object that contains provisioning role definitions.

Table B-4 *Role Definition Object Schema Definition*

Attribute	Description
nrfActive	Whether role is active.
nrfApprovers	Role approvers. Order of approvers is maintained by an integer in the second element.
nrfChildRoles	Child roles of the current role.
nrfEntitlementRef	Identity Manager entitlement associated with the role. Supports embedding of dynamic parameter macros to allow users to specify values at request time.
nrfImplicitContainers	Containers assigned to the role.
	nrfRequestDef Provisioning request definition used for approving a role assignment.
nrfImplicitGroups	Groups assigned to the role.
nrfLocalizedDescrs	The localized description of the role.
nrfLocalizedNames	The localized name of the role.
nrfParentRoles	Parent role of the current role.
nrfQuorum	Used to support quorum approvals in template PRDs. This is the quorum condition. Can be percentage or number of approvers required.
nrfRevokeRequestDef	Provisioning request definition used for approving the revocation of a role assignment.
nrfRoleCategoryKey	Used to categorize role.
nrfRoleLevel	Role level that defines the role hierarchy.
nrfStatus	Status of role. Valid codes are described in"Role Status Codes (nrfStatus)" on page 655.

Role Status Codes (nrfStatus)

Table B-5 *Valid Role (nrfStatus) Status Codes*

Status Code	Key	Description
50	CREATED	Role created.
15	DELETE PENDING	Role delete pending.

Request Object (nrfRequest)

The schema object whose instances contain a role request object. This request object is used by the role driver to provision the role.

Table B-6 *Request Object Schema Definition*

Attribute	Description
nrfApprovalInfo	Holds approval data needed by role view and reports.
nrfApprovalProcessid	Workflow process instance ID for role assignment approval.
nrfApprovers	Role approvers. Order of approvers can be maintained by an integer in the second element.
nrfCategory	10-Role To User Add 15 - Role to User Remove
nrfCorrectionId	Used to group the role assignments request together.
nrfDecisionDate	Indicates date when the request cleanup process evaluation should happen.
nrfDescription	Description/Comment of the role request.
nrfEndDate	Indicates end date of role assignment.
nrfImmediate	Indicates whether the permission has to be assigned immediately.
nrfMacros	Macros definitions for approval by relationship.
nrfOriginator	Used to determine what component originated role assignment request: user application, role request workflow activity, or policy.
nrfQuorum	Used to support quorum approvals in templated PRDs. The quorum condition can be percentage or numbers of approvers required.
nrfRequestDate	Date-time role request started.
nrfRequester	DN of user or role that requested assignment.
nrfRequestDef	Provisioning request definition used for approving the role.
nrfSODApprovalInfo	Approval data needed for SOD violation reporting.

Attribute	Description
nrfSODApprovalProcessId	Provisioning request definition used for SOD Approval if SOD conflict arises.
nrfSODConflicts	List of SOD conflicts with the permission request.
nrfSODQuorum	SOD quorum condition used for resolving SOD conflicts.
nrfSODRequestDef	SOD definition that permission request resulted in conflict.
nrfStartDate	Start date of the role assignment.
nrfSourceDN	DN of user to whom the role is to be added or removed.
nrfTargetDN	DN of user who will be granted the resource or from whom the resource will be revoked.
nrfStatus	Status of request. Valid codes are described in "Request Status Codes (nrfStatus)" on page 656.

Request Status Codes (nrfStatus)

Table B-7 *Valid Request (nrfStatus) Status Codes*

Status Code	Key	Description
00	New Request	Set by User Applications on newly created nrfRequest.
02	SOD RETRY	Driver will reattempt to start the SOD workflow.
03	SOD RETRY PENDING	Occurs when the driver is not able to start a SOD workflow. A driver task will then reset these requests to SOD_WORKFLOW_START_PENDING, to retry the starting of the workflow.
05	SOD PENDING	SOD approval pending; set by the driver after successfully initiating the SOD workflow.
10	SOD APPROVED	SOD approved; set by the SOD workflow when approved.
12	Approval_RETRY	Driver will reattempt to start the workflow.
13	Pending_Approval_RETRY	Occurs when the driver is not able to start the approval workflow.
15	Approval Pending	Set by driver after successful assignment/ revocation workflow.
20	Approved	Set by resource assignment/revocation workflow when approved.

Status Code	Key	Description
25	Assignment PENDING	Activation time pending; set by the driver after obtaining all necessary approvals and when the activation time has not been reached.
30	Provision/Deprovision	Set by driver after all necessary approvals have been approved and role activation time has been reached.
50	Provisioned/Deprovisioned	Set by driver after role has been provisioned or deprovisioned.
70	Cancel	Request cancellation
75	Cancelled	Cancellation request completed.
80	Provisioning Error	Set by driver when an error occurred during provisioning or deprovisioning.
90	SOD Denied	SOD denied; set by SOD exception workflow when denied.
95	DeniedSet	Set by assignment/revocation workflow when approved.
100	CleanupSet	When nrfResourceRequest workflow should be deleted.

Role-Resource Configuration (nrfConfiguration)

Table B-8 *Role-Resource Configuration Object Schema*

Attribute	Definition
nrfResourceRequestContainer	Root container for resource requests.
nrfResourcesContainer	Root container for resource definitions.
nrfResourceRevokeRequestDef	Default PRD for approving resource revocations
nrfResourceGrantRequestDef	Default PRD for approving resource assignments.

Resource Binding to Users (nrfIdentity)

Table B-9 *Resource Binding to Users Object Schema*

Attribute	Description
nrfResource	Currently assigned and assigned resources. Attribute contains DN for the resource DN, the binding state of the resource, and the cause of the assignment and approval information.
nrfResourceHistory	Contains historical information about each resource grant, revocation, denial. Contains the resource as well as XML that contains the resource binding state, (0=inactive, 1=active, 2=pending, 3= deactivated). The XML also contains the entitlement reference value used to grant the entitlement, grant history (who and when), and revocation history (similar to approval information)

Resource Containers

ResourceRequests (nrfResourceRequests): A container objects that persists resource requests.

ResourceDefs (nrfResourceDefs): A container object that persists the definition of a resource.

C JavaScript Search API

The underlying framework for the Identity Manager User Application supports a JavaScript API for executing searches that access the Directory Abstraction Layer. This API lets you build, save, and execute queries from a JSP page running outside of the User Application itself. To run a query, you can invoke the services of the SearchListPortlet, passing parameters that specify the search criteria and formatting options. Alternatively, you can run a search by using the API directly without involving the SearchListPortlet.

Launching a Basic Search using the SearchListPortlet

To perform a basic search, you can specify a *deep link* to the SearchListPortlet from a JSP page. The URL for the portlet must either pass a simple set of request parameters that specify the search criteria, or pass a JSON-formatted query string. A basic search defines a single search criterion, such as the following:

```
First Name starts with A
```

To launch a search, you can call the single portlet render url for the SearchListPortlet. You must pass the request parameter MODE=MODE_RESULTS_LIST

Passing Request Parameters

You can pass a simple set of request parameters to the SearchListPortlet. These parameters specify an entity, an attribute to search on, an operator, and a search string. The following script shows the URL for the portlet, as well as the four request parameters you need to use:

```
<script type="text/javascript">
function openSearchResults(extraUrlParams) {
  var url = "/IDMProv/portal/portlet/SearchListPortlet?";
  url += "urlType=Render&novl-regid=SearchListPortlet";
  url += "&novl-inst=IDMProv.SearchListPortlet";
  url += "&wsrp-mode=view&wsrp-windowstate=normal";
  url += "&MODE=MODE_RESULTS_LIST&";
  url += extraUrlParams;
  var feat = "width=700,height=600";
  feat += ",menubar=no,resizable=yes,toolbar=no,scrollbars=yes";
  var win = window.open(url, "TestSearchPopup", feat);
  if (win) win.focus();
}

var search1a = "ENTITY_DEF=user";
search1a += "&COND_ROW_ATTR=FirstName";
search1a += "&COND_ROW_REL_OP=starts-with";
search1a += "&COND_ROW_VAL=A";
...
```

To call this function, you might have a button on the form with onclick event that looks like this:

```
<input type="button" value="GO" onclick="openSearchResults(search1a)"/>
```

The following table describes the request parameters:

Table C-1 *Request Parameters for Basic Search*

Request Parameter	Description
ENTITY_DEF	Specifies the key value for an entity in the Directory Abstraction Layer.
COND_ROW_ATTR	Specifies the attribute to search on.
COND_ROW_REL_OP	Specifies the operator to use in the search expression. The following operators are supported for attributes of type string, boolean, integer, time, dn_lookup, dynamic_list, and static_list: equals present not_equals not_present The following operators are supported for attributes of type string: starts_with ends_with contains not_starts_with not_ends_with not_contains The following operators are supported for attributes of type integer and time: greater greater_or_equal less less_or_equal not_greater not_greater_or_equal not_less not_less_or_equal
COND_ROW_VAL	The value to search on.

Using a JSON-formatted String to Represent a Query

If you prefer to format your query as a JSON string, you need to pass the QUERY parameter to the SearchListPortlet, instead of the request parameters described in the section above. The JavaScript variable shown below illustrates how the QUERY parameter is constructed:

```
var search1b ='QUERY={"k":"Lastname starts with B","mxPg":"10",';
search1b +='"mxRes":"0","ptr":"1","grp":[{"map":{"row":[{"map":{';
search1b +='"rowRop":"starts-with","rowVal":"B","rowAttr":"LastName"';
search1b +='}}],"rowLop":"and"}}],';
search1b +='"orderBy":"LastName","entDef":"user",';
search1b +='"sScope":"","sRoot":"","grpLop":"and",';
search1b +='"selAttr":["FirstName","LastName",';
search1b +='"Title","Email","TelephoneNumber"]}';
```

The JSON structure gives you a way to specify values for most of the settings and preferences associated with the SearchListPortlet.

The following table describes the JSON name/value pairs that define the QUERY parameter passed to the SearchListPortlet:

Table C-2 *JSON Structure for Defining the QUERY Parameter*

JSON Setting	Description
k	Specifies a name for the search. (Optional)
mxPg	Specifies the maximum number of rows per page. (Optional)
mxRes	Specifies the maximum number of total rows retrieved. (Optional)
ptr	Sets the scroll pointer, which defines the pagination offset. (Optional)
grp	Defines a condition group. You can specify one or more condition groups. For details on the settings for a condition group, see Table C-3 on page 662.
orderBy	Specifies the attribute to sort on. (Optional)
entDef	Specifies an entity in the Directory Abstraction Layer.
sScope	Sets the search scope. (Optional)
sRoot	Sets the search root. (Optional)
grpLop	Defines the logical operator (`and` or `or`) for groups within this query.
selAttr	Lists the attributes to include in the search results.

The following table describes the JSON structure for defining a condition group:

Table C-3 *JSON Structure for Defining a Condition Group*

JSON Setting	Description
row	Defines a condition row. You can specify one or more condition rows. For details on the settings for a condition row, see Table C-4 on page 662.
rowLop	Defines the logical operator (`and` or `or`) for rows within this group.

The following table describes the JSON structure for defining a condition row:

Table C-4 *JSON Structure for Defining the Fields for a Condition Row*

JSON Setting	Description
rowRop	Defines the relational operator. The relational operators supported in JSON are the same as those for basic searches using request parameters. For a complete list of the relational operators, see the description of COND_ROW_REL_OP in Table C-1 on page 660.
rowVal	Sets the search value.
rowAttr	Specifies the attribute to search on.

Creating a New Query using the JavaScript API

As an alternative to using the basic search request parameters, or the JSON structure, you can call a JavaScript API to execute queries. This section describes some simple techniques for using the API, as well as reference documentation for the API.

The search API relies on the ajax framework embedded in the User Application component named JUICE. JUICE (JavaScript UI Controls and Extensions) is compliant with and uses the dojo library. JUICE is merged into the dojo release used in the User Application.

Therefore, to use JUICE on a custom page within the IDM User Application WAR file, you need to have a script reference to dojo.js (not to JUICE). After adding the reference to dojo.js, you can add a JavaScript line to tell dojo to download JUICE.

Before using the JavaScript API, you need to perform some setup steps on the page to make the dojo module available for use:

1. Add a script tag for dojo.js in the HTML header. The reference to dojo.js must be in the header (not the body), as shown below.

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JavaScript Search</title>
<script type="text/javascript">
  if(typeof dojo=="undefined"){
    var djConfig={isDebug: false,
                  baseScriptUri: "/IDMProv/javascript/dojo/"};
    var buf="<script type='text\/javascript' ";
    buf+="src='/IDMProv/UIQuery?js=dojo\/dojo.js'><\/script>";
    document.writeln(buf);
  }
</script>
</head>
```

2 Add this JavaScript statement to load JUICE into the browser's memory:

```
<script type="text/javascript">
  //This line must precede any code using JUICE.
  dojo.require("JUICE.*");
</script>
```

3 To take advantage of the JUICE.IDM services, which include entity searching, also add this JavaScript statement:

```
<script type="text/javascript">
  //This line must precede any code using JUICE.IDM services.
  dojo.require("JUICE.IDM.*");
</script>
```

To build the query, you need to call the create() method on the JUICE.IDM.Entities.Search object, passing in the name you want to give to the query. The create() method is a static method. Here's how you invoke it:

```
var newQuery = JUICE.IDM.Entities.Search.create("My New Search");
```

Once you've created the query object, you can call methods on this object to define the basic settings for the query, as well as the condition groups and condition rows. The query structure you create with the JavaScript API follows the model of the JSON representation. After you've created the query object you append it to the QUERY request parameter.

The JavaScript example shown below illustrates how you use the JavaScript API to build a query:

```
function buildQuery3() {
  var newQuery = JUICE.IDM.Entities.Search.create("My New Search");
  newQuery.setFrom("user");
  var selAttrs = ["FirstName","LastName"];
  newQuery.setSelects(selAttrs);
  var newCondGrp1 = newQuery.addConditionGroup();
  var newCondRow1_1 = newCondGrp1.addConditionRow();
  newCondRow1_1.setRowAttr("FirstName");
  newCondRow1_1.setRowRop("contains");
  newCondRow1_1.setRowVal("C");
  openSearchResults("QUERY=" + newQuery);
}
```

JavaScript API

This section provides reference documentation for the JavaScript API for searching entities in the Directory Abstraction Layer.

The following table describes the static methods for the JUICE.IDM.Entities.Search object:

Table C-5 *Static methods for JUICE.IDM.Entities.Search*

Method	Description
<Query> (createsearchName)	Creates a new Query with the searchName
<void> load(uuid)	Loads a user's saved search with the uuid
<Query> get(uuid)	Returns the user's saved search with uuid as a Query
<String[]> getNames()	Returns the names of all the logged in user's saved searches
<String> getUUID(searchName)	Returns the uuid of the saved search with the searchName

The following table describes the methods for the Query object:

Table C-6 *Methods for the Query object*

Method	Description
<void> setKey(searchName)	Sets the searchName
<void> setFrom(defKey)	Sets the from entity-definition
<void> setSelects(attrKey[])	Sets the selects (optional, if using SearchListPortlet)
<void> setSearchScope(scp)	Sets the search scope (optional)
<void> setSearchRoot(rt)	Sets the search root (optional)
<void> setMaxPage(int)	Sets the max rows per page (optional)
<void> setMaxResults(int)	Sets the max rows in total (optional)
<void> setOrderBy(attrKey)	Sets the sort (optional)
<void> setPointer(int)	Sets the pagination offset (optional)
<void> setGroupLop(lop)	Sets the inter-group logical operator
<String> getKey()	Gets the searchName
<String> getFrom()	Gets the from entity-definition
<String> getSelects()	Gets the selects
<String> getSearchScope()	Gets the search scope
<String> getSearchRoot()	Gets the search root

Method	Description
<int> getMaxPage()	Gets the max rows per page
<int> getMaxResults()	Gets the max rows in total
<String> getOrderBy()	Gets the sort
<int> getPointer()	Gets the pagination offset
<String> getGroupLop()	Gets the inter-group logical operator
<int> nbConditionGroups	Returns the number of condition groups
<CondGroup> addConditionGroup	Creates and returns a new condition group (CondGroup object) appended to the query
<void> removeConditonGroup(i)	Removes the condition group at i
<CondGroup> getConditonGroup(i)	Returns the condition group at i

The following table describes the methods for the CondGroup object:

Table C-7 *Methods for the CondGroup object*

Method	Description
<void> setRowLop(lop)	Sets the intra-group logical operator
<String> getRowLop()	Gets the intra-group logical operator
<int> nbConditionRows()	Returns the number of condition rows
<CondRow> addConditionRow()	Creates and returns a new condition row appended to the condition group
<void> removeConditionRow(i)	Removes the condition row at i
<CondRow> getConditionRow(i)	Returns the condition row at i

The following table describes the methods for the CondRow object:

Table C-8 *Methods for the CondRow object*

Method	Description
<void> setRowAttr(attrKey)	Sets the attribute
<void> setRowRop(rop)	Sets the relational operator.
<void> setRowVal(val)	Sets the search value
<String> getRowAttr()	Gets the attribute
<String> getRowRop()	Gets the relational operator
<String> getRowVal()	Gets the search value

Performing an Advanced Search Using a JSON-formatted Query

You can use the QUERY parameter to perform an advanced search using JSON. The JSON syntax rules are the same as those for the basic search. The only difference is that an advanced search typically defines multiple condition groups and condition rows. The JavaScript variable shown below illustrates how the QUERY parameter might be constructed for a search that uses several condition groups and condition rows:

```
var search2 = 'QUERY={"k":"Complicated Search All
OK","mxPg":"10","mxRes":"0","ptr":"1","grp":[{"map":{"row":[{"map":{"rowRo
p":"equals","rowVal":"cn=bg1,ou=groups,ou=idmsample,o=netiq","rowAttr":"gr
oup"}},{"map":{"rowRop":"contains","rowVal":"0","rowAttr":"FirstName"}}],"
rowLop":"and"}},{"map":{"row":[{"map":{"rowRop":"not-
present","rowVal":"","rowAttr":"TelephoneNumber"}},{"map":{"rowRop":"equal
s","rowVal":"cn=ablake,ou=users,ou=idmsample,o=netiq","rowAttr":"directRep
orts"}},{"map":{"rowRop":"equals","rowVal":"cn=cnano,ou=users,ou=idmsample
,o=netiq","rowAttr":"manager"}}],"rowLop":"and"}},{"map":{"row":[{"map":{"
rowRop":"not-
present","rowVal":"","rowAttr":"TelephoneNumber"}},{"map":{"rowRop":"equal
s","rowVal":"cn=ablake,ou=users,ou=idmsample,o=netiq","rowAttr":"directRep
orts"}},{"map":{"rowRop":"equals","rowVal":"cn=cnano,ou=users,ou=idmsample
,o=netiq","rowAttr":"manager"}}],"rowLop":"and"}}],"orderBy":"LastName","e
ntDef":"user","sScope":"","sRoot":"","grpLop":"or","selAttr":["FirstName",
"Title","Email","TelephoneNumber"]}';
```

For details on each of the JSON settings, see "Using a JSON-formatted String to Represent a Query" on page 660.

Retrieving all Saved Queries for the Current User

You can use the JavaScript API to retrieve all saved queries for the user who is currently logged on. To do this, you need to call the getNames() static method on the JUICE.IDM.Enities.Search object.

The following JavaScript example illustrates the procedure for retrieving all saved queries for the current user:

```
function query4GetSavedQueries() {
  var searchNames = JUICE.IDM.Entities.Search.getNames();
  var replaceDiv = document.getElementById("savedQueryNames");
  replaceDiv.innerHTML = searchNames;
}
```

Running an Existing Saved Query

You can use the JavaScript API to execute a saved query. Before you execute a saved query, you need to perform the following JavaScript statement to retrieve the saved queries (as described in the previous section):

```
JUICE.IDM.Entities.Search.getNames();
```

You need to call getNames() first, even if you know the name of the saved search you want to run.

After calling the getNames() function, you need to perform these steps to execute the saved search:

1. Call the getUUID() method to access the UUID associated with the search name.
2. Call the load() method on the JUICE.IDM.Entities.Search object to load the saved query with the UUID.
3. Call the get() method to retrieve the saved query structure.

All of these methods are static methods.

Once you have the query structure, you can use it to construct a QUERY request parameter.

The following JavaScript example illustrates the procedure for launching a saved query:

```
function runQuery4() {
  var textField = document.getElementById("savedQueryToRun");
  var queryName = textField.value;
  var queryUUID = JUICE.IDM.Entities.Search.getUUID(queryName);
  JUICE.IDM.Entities.Search.load(queryUUID);
  var myQuery = JUICE.IDM.Entities.Search.get(queryUUID);

  openSearchResults("QUERY=" + myQuery);
}
```

Performing a Search on All Searchable Attributes

You can use the JavaScript API to search all of the searchable attributes for an entity. This type of search only applies to attributes that have a type of string. Therefore, it does not work with DN, date, integer, boolean, and so forth.

To perform a search on all searchable attributes, you create a query object in the same manner that you would using other search techniques (as described above). Then you need to get the list of attributes for an entity definition by calling JUICE.IDM.Definition.load(). Once you have the list of attributes, you need to verify that each attribute is a string and is searchable. For each attribute that is a string and is searchable, you can now add a condition row by calling the addConditionRow() method on the condition group object. When all condition rows have been added, you can execute the search.

The following JavaScript example illustrates how to perform a search on all searchable attributes.

```
function buildQuery5() {
  var searchStr = document.getElementById("query5Text").value;
  if (searchStr == "") {
    alert("Enter a search string in the text field.");
    return;
  }
  var newQuery = JUICE.IDM.Entities.Search.create("My New Search");
  var entDef = "user";
  newQuery.setFrom(entDef);
  var selAttrs = new Array();
  selAttrs.push("FirstName");
  selAttrs.push("LastName");
  newQuery.setSelects(selAttrs);
  var newCondGrp1 = newQuery.addConditionGroup();
  newCondGrp1.setRowLop("or");
```

```
  //get all the searchable attributes of entity-definition user that are
type string (excludes DN, date, integer, boolean, etc)
  JUICE.IDM.Definitions.load(entDef);
  var attrKeys = JUICE.IDM.Definitions.getAttributeKeys(entDef);
  for (var i = 0; i < attrKeys.length; i++) {
    var attrDef = JUICE.IDM.Definitions.getAttribute(entDef, attrKeys[i]);
    var attrType = attrDef.getType();
    var searchable = attrDef.isSearchable();

    if (attrType == "String" && searchable ) {
      var newCondRow = newCondGrp1.addConditionRow();
      newCondRow.setRowAttr(attrKeys[i]);
      newCondRow.setRowRop("contains");
      newCondRow.setRowVal(searchStr);
    }
  }
  openSearchResults("QUERY=" + newQuery);
}
```

D Trouble Shooting

This section describes tips for working around common errors.

Permgen Space Error

You might encounter the following error when you redeploy the User Application:

```
11:32:20,194 ERROR [[PortalAggregator]] Servlet.service() for servlet
PortalAggregator threw exception java.lang.OutOfMemoryError: PermGen space
```

To avoid this error, either:

- Restart the Tomcat server.

or

- Or, increase the PermSpace value by passing -XX:MaxPermSize to the Java virtual machine by means of JAVA_OPTS in the start-tomcat script, for example:

  ```
  -XX:MaxpermSize=128m
  ```

Email Notification Templates

If your email notification templates are displaying in a single language and not in the user's default locale as you expect, check to see what notification template is selected. You can select a default template or a localized version of the template. When you select a localized template, the language of the localized template is used regardless of the user's default language. When you select the default template (the template without a locale code), the email is in the user's default language (if the default is a supported language).

Org Chart and Guest Access

If you encounter an error like this at runtime, then you must modify the service definitions in the User Application WAR:

```
error: "an error occurred Control instantiation of JUICE.OrgChartCtrl
failed (Object doesn't support this property or method). Please contact
your system administrator. Detailed information can be found in the
console." when accessing the portlet in a browser.
```

Provisioning Notification

If the **Notify Other Users of these Changes** check box does not display on the following pages:

- Edit Availability
- My Proxy Assignments
- My Delegate Assignments
- Team Proxy Assignments
- Team Delegate Assignments
- Team Availability

Verify that Email Notification templates have been defined. You define them through the **Administration > RBPM Provisioning and Security > Delegation and Proxy**.

javax.naming.SizeLimitExceededException

If you encounter a `javax.naming.SizeLimitExceededException` when you use the **Administration > Page Admin > Set As Default**, you might have encountered a maximum size limit. You can modify this limit in the `PortalGroupPageDefaults` portlet settings in the `portlet.xml` as follows:

```
<portlet>
    <portlet-name>PortalGroupPageDefaults</portlet-name>
   <portlet-class>
com.novell.afw.portal.portlet.core.permission.PortalGroupPageDefaults
</portlet-class>
    <init-param>
      <name>MIN_CACHE_SIZE</name>
      <value>20</value>
    </init-param>
    <init-param>
      <name>MAX_CACHE_SIZE</name>
      <value>200</value>
    </init-param>
    <init-param>
      <name>PAC_MAX_RESULTS</name>
      <value>2000</value>
    </init-param>
    ...
</portlet>
```

If you have more than 200 groups and want to assign groups to the View permissions for the Page Admin tab, you also need to update the settings for the `PortalUserGroupSelection` portlet. Modify this limit in the `portlet.xml` as follows:

```
<portlet>
    <portlet-name>PortalUserGroupSelection</portlet-name>
   <portlet-class>
com.novell.afw.portal.portlet.core.permission.PortalUserGroupSelection
</portlet-class>
    <init-param>
      <name>MIN_CACHE_SIZE</name>
      <value>20</value>
    </init-param>
    <init-param>
      <name>MAX_CACHE_SIZE</name>
      <value>200</value>
    </init-param>
    <init-param>
      <name>PAC_MAX_RESULTS</name>
      <value>2000</value>
    </init-param>
    ...
</portlet>
```

Redeploy the User Application after you make your changes.

Linux Open Files Error

If you run the User Application on Linux, you might encounter a **Too Many Open Files** Error. Linux allows 1024 open files for each process, but the User Application often requires more. NetIQ suggests increasing the number of open files to 4096 to avoid the **Too Many Open Files** error.

Use the `ulimit` command to increase the number of open files. There are some restrictions on `ulimit` for non-root users. Here is an example of how you can use the `ulimit` command to increase the number of open files to 4096 for a non-root user:

1. Log in as root.
2. Edit the file `/etc/security/limits.conf`. Add an entry for the user named **novlua** and allow **nofile** up to 4096:

   ```
   novlua   hard    nofile    4096
   ```

3. Log in as user **novlua** and pass 4096 to the `ulimit -n` command. You can issue the command again with no argument to see the current value:

   ```
   novlua@myhost:~> ulimit -n 4096
   novlua@myhost:~> ulimit -n
   ```

You might want to specify `ulimit` in the user environment or the start-tomcat script so that the new value is always used.