



Samples Guide

/ ForgeRock Identity Management 7

Latest update: 7.0.4

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Copyright © 2011-2021 ForgeRock AS.

Abstract

Guide providing a number of "sample deployments" that walk you through the essential features of ForgeRock® Identity Management software, as they would be implemented.



This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-nd/3.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

ForgeRock® and ForgeRock Identity Platform™ are trademarks of ForgeRock Inc. or its subsidiaries in the U.S. and in other countries. Trademarks are the property of their respective owners.

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

DejaVu Fonts

Bitstream Vera Fonts Copyright

Copyright (c) 2003 by Bitstream, Inc. All Rights Reserved. Bitstream Vera is a trademark of Bitstream, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Bitstream" or the word "Vera".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Bitstream Vera" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL BITSTREAM OR THE GNOME FOUNDATION BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the names of Gnome, the Gnome Foundation, and Bitstream Inc., shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from the Gnome Foundation or Bitstream Inc., respectively. For further information, contact: fonts at gnome dot org.

Arev Fonts Copyright

Copyright (c) 2006 by Tavmjong Bah. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the modifications to the Bitstream Vera Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Tavmjong Bah" or the word "Arev".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Tavmjong Bah Arev" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL TAVMJONG BAH BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the name of Tavmjong Bah shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from Tavmjong Bah. For further information, contact: tavmjong @ free . fr.

FontAwesome Copyright

Copyright (c) 2017 by Dave Gandy, <https://fontawesome.com/>.

This Font Software is licensed under the SIL Open Font License, Version 1.1. See <https://opensource.org/licenses/OFL-1.1>.

Table of Contents

Overview	vi
1. Samples Provided With IDM	1
2. Start Here	6
Run the Samples	6
Prepare IDM	6
LDAP Server Configuration	6
3. Synchronize Data From a CSV File to IDM	11
Sample Overview	11
Sample Configuration Files	11
Run the Sample	14
4. One Way Synchronization From LDAP to IDM	20
Prepare the Sample	20
Run the Sample	20
5. Two Way Synchronization Between LDAP and IDM	24
Prepare the Sample	24
Run the Sample	24
6. Synchronize LDAP Groups	30
Sample Overview	30
Prepare the Sample	30
Run the Sample	31
7. Synchronize LDAP Group Membership	34
Prepare the Sample	34
Run the Sample	35
8. Synchronize Data Between Two External Resources	40
Configure Email for the Sample	40
Run the Sample	41
9. Asynchronous Reconciliation Using a Workflow	44
Run the Sample	44
10. LiveSync With an LDAP Server	48
Set Up the LDAP Resources	49
Run the Sample	50
11. Synchronize Accounts With the Google Apps Connector	54
Prepare the Sample	54
Configure the Google Apps Connector	55
Run the Sample	57
12. Synchronize Users Between Salesforce and IDM	64
Prepare the Sample	64
Run the Sample	64
13. Synchronize Kerberos User Principals	69
Configure the Kerberos Connector	69
Run the Sample	70
14. Store Multiple Passwords For Managed Users	78
Configure the Multiple Passwords Sample	78
Password History Policy	80

LDAP Server Configuration	81
Show Multiple Accounts	81
Show the Password History Policy	85
15. Link Multiple Accounts to a Single Identity	90
Sample Overview	90
Prepare the Sample	92
Run the Sample	92
16. Link Historical Accounts	101
Sample Overview	101
Run the Sample	102
17. Provision Users With Roles	110
Sample Overview	110
Prepare the Sample	110
Run the Sample	112
18. Provision Users With Workflow	131
Prepare the Sample	131
Run the Sample	133
19. Connect to DS With ScriptedREST	136
Set Up DS	136
Run the Sample	138
20. Connect to Active Directory With the PowerShell Connector	147
Sample Overview	147
Prepare the Sample	148
Run the Sample	150
21. Connect to a MySQL Database With ScriptedSQL	157
Configure the External MySQL Database	158
Run the Sample	159
Test the Event Hooks	163
Run the Sample With Paging	166
22. Direct Audit Information To MySQL	168
About the Configuration Files	168
Configure the MySQL Database	169
Run the Sample	170
23. Direct Audit Information to a JMS Broker	173
Dependencies for JMS Messaging	173
Configure SSL for ActiveMQ	174
Configure a Secure Port for JMS Messages	176
Start the ActiveMQ Broker and IDM	176
Configure and Use a JMS Consumer Application	177
24. Synchronize Data Between MongoDB and IDM	180
Sample Overview	180
Configure the MongoDB Database	180
Run the Sample	181
25. Synchronize Data Between IDM and HubSpot	185
Prepare the Sample	185
Run the Sample	186
26. Synchronize Data Between IDM and DocuSign	189

Run the Sample	189
27. Synchronize Data Between IDM and a SCIM Provider	192
Run the Sample	192
28. Subscribe to JMS Messages	197
Sample Overview	197
Dependencies for JMS Messaging	197
Configure SSL for ActiveMQ	199
Configure a Secure Port for JMS Messages	200
Start the ActiveMQ Broker and IDM	200
Access the REST Interface Using the ActiveMQ UI	201
Customize the Scripted JMS Sample	203
29. Authenticate Using a Trusted Servlet Filter	205
Prepare the Sample	205
The Sample Servlet Filter	205
Run the Sample	206
Customize the Sample for an External System	207
30. Create a Custom Endpoint	209
IDM Glossary	213

Overview





This guide describes a number of sample deployments that demonstrate the core functionality of ForgeRock Identity Management software. The samples correspond to the configurations provided in the [openidm/samples](#) directory.

This guide is written for anyone testing ForgeRock Identity Management to manage identities, and to ensure compliance with identity management regulations.

The guide covers a number of ForgeRock Identity Management features, often including multiple features in a single sample.

You don't need a complete understanding of ForgeRock Identity Management software to learn something from this guide, although a background in identity management and maintaining web application software can help. You do need some background in managing services on your operating systems and in your web application containers. You can nevertheless get started with this guide, and then learn more as you go along.

Quick Start

 Overview Get a summary of all the samples in this guide, and learn how to set up a ForgeRock Directory Services (DS) server to run the LDAP samples.	 Start Here Read the setup steps that apply to all samples.	 Sync With CSV Start with a basic sample that shows one-way synchronization from a CSV file to IDM.
 Sync With LDAP Start with a basic sample that shows one-way synchronization from an LDAP directory to IDM.	 Sync Two External Resources Learn how to synchronize two resources without storing data in IDM.	 LiveSync Show <i>liveSync</i> between external LDAP resources.

ForgeRock Identity Platform™ serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see <https://www.forgerock.com>.

The ForgeRock Common REST API works across the platform to provide common ways to access web resources and collections of resources.

Chapter 1

Samples Provided With IDM

This section lists the samples provided with IDM, with a high-level overview of each sample.

A number of samples are provided in the `openidm/samples` directory:

Getting Started

The Getting Started sample describes how to install and evaluate IDM.

`(samples/getting-started)`

"Synchronize Data From a CSV File to IDM"

This sample demonstrates one-way synchronization from an external resource to an IDM repository. The external resource in this case is a simple CSV file. User objects in that file are synchronized with the managed users in the IDM repository.

`(samples/sync-with-csv)`

"One Way Synchronization From LDAP to IDM"

This sample uses the generic LDAP connector to connect to an LDAP directory. The sample includes one mapping from the LDAP directory to the managed user repository, and demonstrates reconciliation from the external resource to the repository.

`(samples/sync-with-ldap)`

"Two Way Synchronization Between LDAP and IDM"

This sample uses the generic LDAP connector to connect to an LDAP directory. The sample includes two mappings - one from the LDAP directory to the managed user repository, and one in the opposite direction. The sample demonstrates reconciliation from the LDAP directory to the repository and implicit synchronization from the managed user repository to the LDAP directory.

`(samples/sync-with-ldap-bidirectional)`

"Synchronize LDAP Groups"

This sample uses the generic LDAP connector to connect to an LDAP directory. The sample builds on the previous sample by providing an additional mapping, from the LDAP groups object, to the managed groups object. The sample illustrates a new managed object type (groups) and shows how this object type is synchronized with group containers in LDAP.

(samples/sync-with-ldap-groups)

"Synchronize LDAP Group Membership"

This sample uses the generic LDAP connector to connect to an LDAP directory. The sample includes two mappings, one from the LDAP directory to the managed user repository, and one from the repository to the LDAP directory. The sample demonstrates synchronization of group membership, that is, how the value of the `ldapGroups` property in a managed user object is mapped to the corresponding user object in LDAP.

(samples/sync-with-ldap-group-membership)

"Synchronize Data Between Two External Resources"

This sample demonstrates synchronization between two external resources, routed through the IDM repository. The resources are named `LDAP` and `AD` and represent two separate LDAP directories. In the sample both resources are simulated with simple CSV files.

(samples/sync-two-external-resources)

"Asynchronous Reconciliation Using a Workflow"

This sample shows how you can use workflows to launch an asynchronous reconciliation operation.

(samples/sync-asynchronous)

"LiveSync With an LDAP Server"

This sample illustrates the liveSync mechanism that pushes changes from an external resource to the IDM repository. The sample uses an LDAP connector to connect to an LDAP directory, either ForgeRock Directory Services (DS) or Active Directory.

(samples/livesync-with-ad)

"Synchronize Accounts With the Google Apps Connector"

This sample uses the Google Apps Connector to create users and groups on an external Google system and to reconcile those accounts with the IDM managed user repository.

(samples/sync-with-google)

"Synchronize Users Between Salesforce and IDM"

This sample demonstrates how to create and update users in Salesforce, using the Salesforce Connector. The sample also shows synchronization of users between Salesforce and the IDM managed user repository.

(samples/sync-with-salesforce)

"Synchronize Kerberos User Principals"

This sample demonstrates how to use the scripted Kerberos connector to manage Kerberos user principals and to reconcile user principals with IDM managed user objects.

([samples/sync-with-kerberos](#))

"Store Multiple Passwords For Managed Users"

This sample demonstrates how to set up multiple passwords for managed users and how to synchronize separate passwords to different external resources. The sample includes two target LDAP servers, each with different password policy and encryption requirements. The sample also shows how to extend the password history policy to apply to multiple password fields.

([samples/multiple-passwords](#))

"Link Multiple Accounts to a Single Identity"

This sample illustrates how IDM addresses links from multiple accounts to one identity. The sample shows how you can create links between a single source account and multiple target accounts, using *link qualifiers* that enable one-to-many relationships in mappings and policies.

([samples/multi-account-linking](#))

"Link Historical Accounts"

This sample demonstrates the retention of inactive (historical) LDAP accounts that have been linked to a corresponding managed user account.

([samples/historical-account-linking](#))

"Provision Users With Roles"

This sample builds on the sample described in "*One Way Synchronization From LDAP to IDM*", and demonstrates how attributes are provisioned to an external system (an LDAP directory), based on role membership.

([samples/provisioning-with-roles](#))

"Provision Users With Workflow"

The provisioning workflow sample demonstrates a typical use case of a workflow — provisioning new users. The sample demonstrates the use of the End User UI to allow users to complete a registration process.

([samples/provisioning-with-workflow](#))

"Connect to DS With ScriptedREST"

This sample uses the Groovy Connector Toolkit to implement a ScriptedREST connector that interacts with the DS REST API.

(samples/scripted-rest-with-dj)

"Connect to a MySQL Database With ScriptedSQL"

This sample uses the Groovy Connector Toolkit to implement a ScriptedSQL connector that interacts with an external MySQL database.

(samples/scripted-sql-with-mysql)

"Connect to Active Directory With the PowerShell Connector"

This sample uses the MS Active Directory PowerShell module to demonstrate how you can synchronize managed objects with a Microsoft Active Directory deployment. The sample provides a number of PowerShell scripts that enable you to perform basic CRUD (create, read, update, delete) operations on an Active Directory server.

(samples/scripted-powershell-with-ad)

"Direct Audit Information To MySQL"

This sample uses a ScriptedSQL implementation of the Groovy Connector Toolkit to direct audit information to a MySQL database.

(samples/audit-jdbc)

"Direct Audit Information to a JMS Broker"

This sample demonstrates how the JMS audit event handler can publish messages that comply with the *Java(TM) Message Service Specification Final Release 1.1*

(samples/audit-jms)

"Synchronize Data Between MongoDB and IDM"

This sample uses the Groovy Connector Toolkit to implement a scripted connector that interacts with a MongoDB Database. The connector can be used for provisioning MongoDB database users and roles from an IDM managed repository.

(samples/sync-with-mongodb)

"Synchronize Data Between IDM and HubSpot"

This sample demonstrates bidirectional synchronization between IDM managed users and HubSpot contacts.

(samples/sync-with-hubspot)

"Synchronize Data Between IDM and DocuSign"

This sample demonstrates bidirectional synchronization between IDM managed users and DocuSign user accounts.

(samples/sync-with-docusign)

"Synchronize Data Between IDM and a SCIM Provider"

This sample demonstrates bidirectional synchronization between IDM managed users and roles with corresponding users and roles from a SCIM provider.

(samples/sync-with-scim)

"Subscribe to JMS Messages"

This sample demonstrates the scripted JMS message handler, and how it performs ForgeRock REST operations.

(samples/scripted-jms-subscriber)

"Authenticate Using a Trusted Servlet Filter"

This sample demonstrates how to use a custom servlet filter and the "Trusted Request Attribute Authentication Module" to allow IDM to authenticate through another service.

(samples/trusted-servlet-filter)

"Create a Custom Endpoint"

IDM supports scriptable custom endpoints that enable you to launch arbitrary scripts through an IDM REST URI. This example shows how custom endpoints are configured and returns a list of variables available to each method used in a custom endpoint script.

(samples/example-configurations/custom-endpoint)

Example Configuration Files

In addition to these samples, IDM provides example configuration and data files that you can use to set up your own project. These files are in the `samples/example-configurations` directory. Details on each of these files is provided in the documentation that corresponds to the purpose of the file. For example, the `conf/external.email.json` file is described in "Configure Outbound Email" in the *External Services Guide*.

Chapter 2

Start Here

Before you try any of the samples read "Run the Samples" and "Prepare IDM".

Read "LDAP Server Configuration" for any samples that require an LDAP server.

Run the Samples

Each sample directory in `openidm/samples/` contains a number of subdirectories, such as `conf/` and `script/`. To start IDM with a sample configuration, navigate to the `/path/to/openidm` directory and use the `-p` option of the **startup** command to point to the sample whose configuration you want to use. Some samples require additional software, such as an external LDAP server or database.

Many of the procedures in this guide refer to paths such as `samples/sample-name`. In each of these cases, the complete path is assumed to be `/path/to/openidm/samples/sample-name`.

When you move from one sample to the next, you are changing the IDM configuration. For more information, see "Configuration Changes" in the *Setup Guide*.

The command-line examples in the IDM documentation assume a UNIX shell. To run the samples on Windows, adjust the commands, as necessary.

Prepare IDM

Install an instance of IDM specifically to experiment with the samples and easily discard the result when you finish.

If you are using the same IDM instance for multiple samples, clear the repository between samples. To do so, shut down IDM and delete the `openidm/db/openidm` directory:

```
rm -rf /path/to/openidm/db/openidm
```

LDAP Server Configuration

For samples in this guide that require an LDAP server, ForgeRock recommends using ForgeRock Directory Services (DS).

+ Sample LDAP Server Configuration

- The LDAP server runs on the local host.
- The LDAP server listens on port 1389.
- The replication port is 8989.

Servers with replication ports maintain a changelog for their own use. The changelog is exposed over LDAP under the base DN, `cn=changelog`. For samples that demonstrate liveSync with an LDAP server, you *must* configure a replication port when you set up DS. For ease of use, all the LDAP samples assume that you have configured a replication port, even if you don't use liveSync.

- A user with DN `uid=admin` and password `password` has read access to the LDAP server.
- Directory data for that server is stored under base DN `dc=com`.
- User objects for that server are stored under base DN `ou=People,dc=example,dc=com`.
- User objects have the object class `inetOrgPerson`.
- User objects have the following attributes:

- `cn`
- `description`
- `givenName`
- `mail`
- `sn`
- `telephoneNumber`
- `uid`
- `userPassword`

+ Example User Object

```
dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
givenName: Barbara
uid: bjensen
cn: Barbara Jensen
telephoneNumber: 1-360-229-7105
sn: Jensen
mail: bjensen@example.com
description: Created for OpenIDM
userPassword: password
```

Note

If you are using the same DS instance for multiple samples, delete the DS configuration between samples:

1. Shutdown DS:

```
/path/to/openssl/bin/stop-ds --quiet
```

2. Delete the `openssl/db` directory:

```
rm -rf /path/to/openssl/db
```

3. Delete the `openssl/config` directory:

```
rm -rf /path/to/openssl/config
```

Start DS Using Sample LDIF data

Samples that use an LDAP server require existing user data. The example procedure below corresponds to the `sync-with-ldap` sample and imports user data (`openidm/samples/sync-with-ldap/data/Example.ldif`) during DS setup. For other samples, replace the path to the sample data, as necessary.

Note

The following procedure provides setup instructions for DS 7. For older versions of DS, or an alternative LDAP server, modify the instructions, as necessary.

1. Download the DS and IDM .zip archives.
2. Extract the .zip archives.
3. Start DS:

```
/path/to/openssl/setup \
--serverId evaluation-only \
--deploymentKeyPassword password \
```

```
--rootUserDN uid=admin \
--rootUserPassword password \
--hostname localhost \
--adminConnectorPort 4444 \
--ldapPort 1389 \
--enableStartTls \
--ldapsPort 1636 \
--replicationPort 8989 \
--httpPort 8090 \
--profile ds-user-data:7.0.0 \
--set ds-user-data/baseDn:dc=com \
--set ds-user-data/ldifFile:/path/to/openidm/samples/sync-with-ldap/data/Example.ldif \
--acceptLicense \
--start

Validating parameters..... Done
Configuring certificates..... Done

Store the following deployment key in a safe place and re-use it when
configuring other servers in the topology:

your-deployment-key

Configuring server..... Done
Configuring profile DS application data store..... Done
Starting directory server..... Done

To see basic server status and configuration, you can launch
/path/to/opendj/bin/status
```

Note

Every DS deployment requires a *deployment key* and a *deployment key password* to secure network connections. The deployment key is a random string generated by DS software. The deployment key password is a secret string that you choose. It must be at least 8 characters long. The deployment key and password automate key pair generation and signing without storing the CA private key. For more information, see Deployment Keys in the *DS Security Guide*.

4. Import the DS CA certificate into the IDM truststore:

```
/path/to/opendj/bin/dskeymgr \
export-ca-cert \
--deploymentKey your-deployment-key \
--deploymentKeyPassword password \
--alias dscert \
--keyStoreFile /path/to/openidm/security/truststore \
--keyStorePasswordFile /path/to/openidm/security/storepass
```

Note

Because each new deployment of DS has a unique deployment key, the same certificate does not work from one sample to the next. To handle this scenario, do one of the following:

- Give each subsequent sample certificate a unique alias. For example:

- `--alias dscert1`
- `--alias dscert2`
- `--alias dscert3`
- Delete the old certificate from the trust store:

```
keytool \  
-delete \  
-keystore /path/to/openidm/security/truststore \  
-alias dscert
```

Chapter 3

Synchronize Data From a CSV File to IDM

This sample demonstrates one-way synchronization from an external resource to an IDM repository.

The external resource in this case is a simple CSV file. User objects in that file are synchronized with the managed users in the IDM repository.

Sample Overview

IDM connects data objects held in separate resources by mapping one object to another. To connect to external resources, IDM uses *connectors*, that are configured for each external resource.

When objects in one external resource change, IDM determines how the changes affect the objects in the connected resource, and can make the changes in that resource as necessary. This sample demonstrates how IDM does this by using *reconciliation*. Reconciliation compares the objects in one resource to the mapped objects in another resource. For a complete explanation of reconciliation and synchronization, see "Types of Synchronization" in the *Synchronization Guide*.

In this sample, IDM connects to a CSV file that holds sample user data. The CSV file is configured as the authoritative source. A *mapping* is configured between objects in the CSV file and managed user objects in the IDM repository.

Note that you can use IDM to synchronized objects between two external resources without going through the IDM repository. In such a case, objects are synchronized directly through connectors to the external resources.

This sample involves only one external resource. In practice, you can connect as many resources as needed for your deployment.

Sample Configuration Files

The configuration files for this sample are located in the `/path/to/openidm/samples/sync-with-csv/conf` directory. When you start IDM with the `-p` project variable (`./startup.sh -p samples/sync-with-csv`), the *project location* (`&{idm.instance.dir}`) is set to a value of `samples/sync-with-csv`. All subsequent paths use this project location as a base. Throughout this documentation, you will see things like "...in your project's `conf/` directory...". The project here refers to the the value of the `&{idm.instance.dir}` variable.

The following configuration files play important roles in this sample:

`samples/sync-with-csv/conf/provisioner.openicf-csvfile.json`

This file provides the configuration for this instance of the CSV connector. It describes, among other things, the connector version, the location of the CSV file resource, and the object types that are supported for this connection. For a complete understanding of connector configuration files, see "[Configure Connectors](#)" in the *Connectors Guide*.

`samples/sync-with-csv/conf/sync.json`

This file, also called a *mapping file*, defines the configuration for reconciliation and synchronization. This sample file includes only one mapping - `systemCsvfileAccounts_managedUser`. The mapping specifies the synchronization configuration between the CSV file (source) and the IDM repository (target). Examine the file to see how objects are mapped between the two resources, and the actions that IDM should take when it finds objects in specific situations:

```
{
  "mappings": [
    {
      "name": "systemCsvfileAccounts_managedUser",
      "source": "system/csvfile/account",
      "target": "managed/user",
      "correlationQuery": {
        "type": "text/javascript",
        "source": "var query = {'_queryId' : 'for-username',
          'uid' : source.name};query;"
      },
      "properties": [
        {
          "source": "email",
          "target": "mail"
        },
        {
          "source": "firstname",
          "target": "givenName"
        },
        {
          "source": "lastname",
          "target": "sn"
        },
        {
          "source": "description",
          "target": "description"
        },
        {
          "source": "_id",
          "target": "_id"
        },
        {
          "source": "name",
          "target": "userName"
        },
        {
          "source": "password",
          "target": "password"
        }
      ]
    }
  ]
}
```

```

        "source" : "mobileTelephoneNumber",
        "target" : "telephoneNumber"
    },
    {
        "source" : "roles",
        "transform" : {
            "type" : "text/javascript",
            "source" : "var _ = require('lib/lodash'); _.map(source.split(','),
                function(role) { return {'_ref': 'internal/role/' + role} });"
        },
        "target" : "authzRoles"
    }
}
],
"policies": [
    {
        "situation": "CONFIRMED",
        "action": "UPDATE"
    },
    {
        "situation": "FOUND",
        "action": "IGNORE"
    },
    {
        "situation": "ABSENT",
        "action": "CREATE"
    },
    {
        "situation": "AMBIGUOUS",
        "action": "IGNORE"
    },
    {
        "situation": "MISSING",
        "action": "IGNORE"
    },
    {
        "situation": "SOURCE_MISSING",
        "action": "IGNORE"
    },
    {
        "situation": "UNQUALIFIED",
        "action": "IGNORE"
    },
    {
        "situation": "UNASSIGNED",
        "action": "IGNORE"
    }
]
}

```

Source and target paths that start with **managed**, such as **managed/user**, always refer to objects in the IDM repository. Paths that start with **system**, such as **system/csvfile/account**, refer to external objects, in this case, objects in the CSV file.

When you start a reconciliation, IDM queries all users in the source, and then creates, deletes, or modifies users in the IDM repository, as mapped in **conf/sync.json**.

For more information about synchronization, reconciliation, and mappings, see the [Synchronization Guide](#).

`samples/sync-with-csv/conf/schedule-reconcile_systemCsvAccounts_managedUser.json`

The sample schedule configuration file defines a task that launches a reconciliation every minute for the mapping named `systemCsvfileAccounts_managedUser`. The schedule is disabled by default:

```
{
  "enabled" : false,
  "type": "simple",
  "repeatInterval": 3600000,
  "persisted" : true,
  "concurrentExecution" : false,
  "misfirePolicy" : "fireAndProceed",
  "invokeService" : "sync",
  "invokeContext" : {
    "action" : "reconcile",
    "mapping" : "systemCsvfileAccounts_managedUser"
  }
}
```

IDM regularly scans the `conf/` directory for any `schedule configuration` files.

Apart from the scheduled reconciliation run, you can also start reconciliation run through the REST interface. The call to the REST interface is an HTTP POST such as the following:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemCsvfileAccounts_managedUser&waitForCompletion=true"
```

The `waitForCompletion=true` parameter specifies that the operation should return only when it has completed.

`samples/sync-with-csv/data/csvConnectorData.csv`

This CSV file is the external resource or data store in this sample. The file contains two users, bjensen and scarter. During the sample, you will reconcile those users *from* the CSV file *to* the managed user repository.

Run the Sample

To run this sample, start IDM with the configuration for the sample:

```
/path/to/openidm/startup.sh -p samples/sync-with-csv
```

You can work through the sample using the command line, or using the Admin UI:

+ Use the Command Line

1. When you have started IDM, reconcile the objects in both resources.

You can trigger the reconciliation either by setting `"enabled" : true` in the schedule configuration file (`conf/schedule-reconcile_systemCsvAccounts_managedUser.json`) and then waiting until the scheduled reconciliation happens, or by running the following **curl** command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemCsvfileAccounts_managedUser&waitForCompletion=true"
```

Successful reconciliation returns a reconciliation run ID, and the status of the reconciliation operation, as follows:

```
{
  "_id": "2d87c817-3d00-4776-a705-7de2c65937d8",
  "state": "SUCCESS"
}
```

2. Display the managed user records that were created by the reconciliation operation.

You can use any REST client to query the repository. Perform an HTTP GET on the URL `http://localhost:8080/openidm/managed/user?_queryFilter=true` with the headers `"X-OpenIDM-Username: openidm-admin"` and `"X-OpenIDM-Password: openidm-admin"`. The following example uses the **curl** command to get all managed user records, in JSON format:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true"

{
  "result": [
    {
      "_id": "bjensen",
      "_rev": "00000000e17186b6",
      "mail": "bjensen@example.com",
      "givenName": "Barbara",
      "sn": "Jensen",
      "description": "Created By CSV",
      "userName": "bjensen",
      "telephoneNumber": "1234567",
      "accountStatus": "active",
      "effectiveAssignments": [],
      "effectiveRoles": []
    },
    {
```

```
{
  "_id": "scarter",
  "_rev": "00000000970685c3",
  "mail": "scarter@example.com",
  "givenName": "Steven",
  "sn": "Carter",
  "description": "Created By CSV",
  "userName": "scarter",
  "telephoneNumber": "1234567",
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": []
},
...
}
```

You can use any query filter to return the information you need. For more information, see ["Define and Call Data Queries"](#) in the *Object Modeling Guide*.

- Now display user bjensen's record by appending her user ID to the URL:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/bjensen"
{
  "_id": "bjensen",
  "_rev": "00000000e17186b6",
  "mail": "bjensen@example.com",
  "givenName": "Barbara",
  "sn": "Jensen",
  "description": "Created By CSV",
  "userName": "bjensen",
  "telephoneNumber": "1234567",
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": []
}
```

This command returns bjensen's complete user record.

- Restrict the query output with the `fields` parameter, as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true&_fields=username,mail"
{
  "result": [
    {
      "_id": "bjensen",
      "_rev": "00000000e17186b6",
      "mail": "bjensen@example.com",
      "userName": "bjensen"
    },
    {
      "_id": "scarter",
      "_rev": "00000000970685c3",
      "mail": "scarter@example.com",
      "userName": "scarter"
    }
  ],
  ...
}
```

5. To test the scheduled reconciliation, add a user to the CSV data file, [samples/sync-with-csv/data/csvConnectorData.csv](#). For example, add user jberg as follows:

```
"description", "uid", "username", "firstname", "lastname", "email", "mobile..."
"Created ...", "bjensen", "bjensen@example.com", "Barbara", "Jensen", "bjensen@example.com",
"123456..."
"Created ...", "scarter", "scarter@example.com", "Steven", "Carter", "scarter@example.com",
"123456..."
"Created ...", "jberg", "jberg@example.com", "James", "Berg", "jberg@example.com", "123456..."
```

6. If you enabled the scheduled reconciliation in Step 1, you can simply wait for the reconciliation operation to run. Otherwise, run the reconciliation manually with the same command you used in that step.
7. After the reconciliation has run, query the managed user repository to see the new user in the list of managed users:


```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true&_fields=_id"
{
  "result": [
    {
      "_id": "bjensen",
      "_rev": "00000000e17186b6"
    },
    {
      "_id": "scarter",
      "_rev": "00000000970685c3"
    },
    {
      "_id": "jberg",
      "_rev": "00000000ea628233"
    }
  ],
  ...
}
```

8. To view the reconciliation details, query the reconciliation using its **id**:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/recon/assoc/2d87c817-3d00-4776-a705-7de2c65937d8"
{
  "_id": "2d87c817-3d00-4776-a705-7de2c65937d8",
  "_rev": "1",
  "sourceResourceCollection": "managed/user",
  "targetResourceCollection": "system/csv/account",
  "isAnalysis": "false",
  "finishTime": "2022-05-01T23:36:24.434153Z"
}
```

Note

If you've enabled audit logging, you can view the reconciliation details in the `openidm/audit/recon.audit.json` file.

You configure the action that IDM takes for each situation in the mapping file, `conf/sync.json`. For the list of all possible situations and actions, see "*Synchronization Situations and Actions*" in the *Synchronization Guide*.

+ Use the Admin UI

IDM includes a browser-based Administrative User Interface, known as the Admin UI. For details, see "*Admin UI*" in the *Setup Guide*.

After starting IDM, access the Admin UI by navigating to <https://localhost:8443/admin>. The first time you log in, use the default administrative credentials, (Login: openidm-admin, Password: openidm-admin).

Warning

To protect your deployment in production, change the default administrative password, as described in "Change the Administrator User Password" in the *Security Guide*.

You should now see the Dashboard screen, with quick start cards for common administrative tasks, with the connectors and managed objects associated with that configuration.

1. Reconcile the two resources as follows:

Click Configure > Mappings, select the `systemCsvfileAccounts_managedUser` mapping, and click Reconcile.

2. After reconciliation, display the user records in both the source and target resources.

Select the Association tab and scroll down to the bottom of the page to see the resulting source and target users.

Chapter 4

One Way Synchronization From LDAP to IDM

This sample demonstrates one-way synchronization from an LDAP directory to an IDM repository and shows how IDM detects new or changed objects from an external resource.

The sample has been tested with ForgeRock Directory Services (DS) but should work with any LDAPv3-compliant server. The configuration includes one mapping, from the LDAP resource to the IDM repository. The sample does not push any changes made to IDM managed user objects out to the LDAP server.

The mapping configuration file (`conf/sync.json`) for this sample includes one mapping, `systemLdapAccounts_managedUser`, which synchronize users from the source LDAP server with the target IDM repository.

Prepare the Sample

1. Set up DS using `/path/to/openidm/samples/sync-with-ldap/data/Example.ldif`.
2. Prepare IDM, and start the server using the sample configuration:

```
cd /path/to/openidm/  
./startup.sh -p samples/sync-with-ldap
```

Run the Sample

You can work through the sample using the command line or Admin UI:

+ *Use the Command Line*

1. Reconcile the repository:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemLdapAccounts_managedUser&waitForCompletion=true"
{
  "_id": "b1394d10-29b0-4ccf-81d8-c88948ea121c-4",
  "state": "SUCCESS"
}
```

The reconciliation operation creates the two users from the LDAP server in the IDM repository and assigns the new objects random unique IDs.

2. Retrieve the users from the repository:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true&_fields=id,userName"
{
  "result": [
    {
      "_id": "0326cbff-8f6e-4531-97dd-7b1a4c04b23a",
      "_rev": "00000000657c9a27",
      "userName": "bjensen"
    },
    {
      "_id": "9afbf2bc-0323-4cbe-89b3-92f2f47742c3",
      "_rev": "0000000015ae92f5",
      "userName": "jdoe"
    }
  ],
  ...
}
```

3. To retrieve an individual user object, include their ID in the URL. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/0326cbff-8f6e-4531-97dd-7b1a4c04b23a"
{
  "_id": "0326cbff-8f6e-4531-97dd-7b1a4c04b23a",
  "_rev": "00000000657c9a27",
  "displayName": "Barbara Jensen",
  "description": "Created for OpenIDM",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "sn": "Jensen",
  "telephoneNumber": "1-360-229-7105",
  "userName": "bjensen",
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": []
}
```

+ Use the Admin UI

1. Log in to the Admin UI at <http://localhost:8080/admin> as the default administrative user: `openidm-admin` with password `openidm-admin`.

Warning

To protect your deployment in production, you must change the default administrative password. For more information, see ["Change the Administrator User Password"](#) in the *Security Guide*.

2. Select Configure > Mappings.

The Mappings page displays one mapping, from the `Ldap` server to the IDM repository (`Managed/ User`).

3. Select the mapping, and click Reconcile.

The reconciliation operation creates the two users from the LDAP server in the IDM repository.

4. To verify the new users exist in the repository:

1. From the navigation bar, click Manage > User.

IDM displays the two users.

2. To view the details for a user account, from the User List page, click any username row.

The User details page displays.

Chapter 5

Two Way Synchronization Between LDAP and IDM

This sample demonstrates bidirectional synchronization between an LDAP directory and an IDM repository.

The sample has been tested with ForgeRock Directory Services, but should work with any LDAPv3-compliant server. The configuration includes two mappings, one from the LDAP resource to the IDM repository, and one from IDM to LDAP.

In this sample, you will start IDM and reconcile the two data sources. The mapping configuration file (`sync.json`) for this sample includes two mappings, `systemLdapAccounts_managedUser`, which synchronizes users from the source LDAP server with the target repository, and `managedUser_systemLdapAccounts`, which synchronizes changes from the repository to the LDAP server.

Prepare the Sample

1. Set up DS using `/path/to/openidm/samples/sync-with-ldap-bidirectional/data/Example.ldif`.
2. Prepare IDM, and start the server using the sample configuration:

```
cd /path/to/openidm/  
./startup.sh -p samples/sync-with-ldap-bidirectional
```

Run the Sample

You can work through the sample using the command line or the Admin UI:

+ *Use the Command Line*

1. Reconcile the repository over the REST interface:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemLdapAccounts_managedUser&waitForCompletion=true"
{
  "state": "SUCCESS",
  "_id": "027e25e3-7a33-4858-9080-161c2b40a6bf-2"
}
```

The reconciliation operation returns a reconciliation run ID and the status of the operation. Reconciliation creates user objects from LDAP in the IDM repository, assigning the new objects random unique IDs.

2. To retrieve the users from the repository, query their IDs:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true&_fields=_id"
{
  "result": [
    {
      "_id": "d460ed00-74f9-48fb-8cc1-7829be60ddac",
      "_rev": "00000000792afa08"
    },
    {
      "_id": "74fe2d25-4eb1-4148-a3ae-ff80f194b3a6",
      "_rev": "00000000a92657c7"
    }
  ],
  ...
}
```

3. To retrieve individual user objects, include the ID in the URL, for example:


```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/d460ed00-74f9-48fb-8cc1-7829be60ddac"
{
  "_id": "d460ed00-74f9-48fb-8cc1-7829be60ddac",
  "_rev": "00000000792afa08",
  "displayName": "Barbara Jensen",
  "description": "Created for OpenIDM",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "telephoneNumber": "1-360-229-7105",
  "sn": "Jensen",
  "userName": "bjensen",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

4. To test the second mapping, create a user in the IDM repository:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "mail": "fdoe@example.com",
  "sn": "Doe",
  "telephoneNumber": "555-1234",
  "userName": "fdoe",
  "givenName": "Felicitas",
  "description": "Felicitas Doe",
  "displayName": "fdoe"}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "_id": "90d1f388-d8c3-4438-893c-be4e498e7a1c",
  "_rev": "00000000792afa08",
  "mail": "fdoe@example.com",
  "sn": "Doe",
  "telephoneNumber": "555-1234",
  "userName": "fdoe",
  "givenName": "Felicitas",
  "description": "Felicitas Doe",
  "displayName": "fdoe",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

5. By default, *implicit synchronization* is enabled for mappings from the `managed/user` repository to any external resource. This means that when you update a managed object, any mappings defined in the `sync.json` file that have the managed object as the source are automatically

executed to update the target system. For more information, see "*Mapping Data Between Resources*" in the *Synchronization Guide*.

To test that the implicit synchronization has been successful, query the users in the LDAP directory over REST:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account?_queryId=query-all-ids"
{
  "result": [
    {
      "_id": "0da50512-79bb-3461-bd04-241ee4c785bf"
    },
    {
      "_id": "887732e8-3db2-31bb-b329-20cd6fcecc05"
    },
    {
      "_id": "2f03e095-ec81-4eb5-9201-a4df2f1f9add"
    }
  ],
  ...
}
```

Note the additional user entry.

6. To query the complete entry, include the `_id` in the URL:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account/2f03e095-ec81-4eb5-9201-a4df2f1f9add"
{
  "_id": "2f03e095-ec81-4eb5-9201-a4df2f1f9add",
  "givenName": "Felicitas",
  "dn": "uid=fdoe,ou=People,dc=example,dc=com",
  "mail": "fdoe@example.com",
  "ldapGroups": [],
  "uid": "fdoe",
  "employeeType": [],
  "aliasList": [],
  "telephoneNumber": "555-1234",
  "kbaInfo": [],
  "cn": "fdoe",
  "objectClass": [
    "person",
    "organizationalPerson",
    "inetOrgPerson",
    "top"
  ],
  "sn": "Doe",
  "description": "Felicitas Doe"
}
```

+ Use the Admin UI

1. Log in to the Admin UI.
2. From the navigation bar, click Configure > Mappings.

The Mappings page displays two configured mappings, one from the **ldap** server to the IDM repository (**managed/user**) and one from the repository to the **ldap** server.

3. Select the LDAP to managed user mapping, and click Reconcile.

The reconciliation operation creates the two users from the LDAP server in the IDM repository.

4. To view the new users in the repository, from the navigation bar, click Manage > User.

IDM displays the two users.

5. To add a user account, from the User List page, click + New User.
6. On the New User page, enter the user details, and click Save.

Note

By default, *implicit synchronization* is enabled for mappings *from* the **managed/user** repository *to* any external resource. This means that when you update a managed object, any mappings defined in the **sync.json** file that have the managed object as the source are automatically executed to update the target system. For more information, see "*Mapping Data Between Resources*" in the *Synchronization Guide*.

7. To test for successful implicit synchronization, from the navigation bar, click Manage > User.
 - a. From the Users List page, click the new user you created in the previous step.
 - b. Click the Linked Systems tab.

IDM displays the user's mapped external resource.

Chapter 6

Synchronize LDAP Groups

This sample demonstrates synchronization between an LDAP directory and an IDM repository. The sample synchronizes LDAP group objects (rather than LDAP group membership, demonstrated in "*Synchronize LDAP Group Membership*").

The sample has been tested with ForgeRock Directory Services (DS) but should work with any LDAPv3-compliant server. The sample includes mappings from the LDAP server to the IDM repository, and from the IDM repository to the LDAP server. During reconciliation, user entries and group entries are synchronized.

Sample Overview

The mapping configuration file, `conf/sync.json`, for this sample includes three mappings:

`systemLdapAccounts_managedUser`

Synchronizes users from the source LDAP server with the target IDM repository.

`managedUser_systemLdapAccounts`

Synchronizes users from the IDM repository to the LDAP server.

`systemLdapGroups_managedGroup`

Synchronizes groups from the source LDAP server with the target IDM repository.

This sample focuses only on the groups mapping, `systemLdapGroups_managedGroup`.

Prepare the Sample

1. Set up DS using `/path/to/openidm/samples/sync-with-ldap-groups/data/Example.ldif`.

The import file includes a number of LDAP groups, including:

```
dn: ou=Groups,dc=example,dc=com
ou: Groups
objectClass: organizationalUnit
objectClass: top

dn: cn=openidm,ou=Groups,dc=example,dc=com
uniqueMember: uid=jdoe,ou=People,dc=example,dc=com
cn: openidm
objectClass: groupOfUniqueNames
objectClass: top

dn: cn=openidm2,ou=Groups,dc=example,dc=com
uniqueMember: uid=bjensen,ou=People,dc=example,dc=com
cn: openidm2
objectClass: groupOfUniqueNames
objectClass: top
```

The user with dn `uid=jdoe,ou=People,dc=example,dc=com` is also imported with the `Example.ldif` file.

There is an additional user, `bjensen` in the sample LDIF file. This user is essentially a "dummy" user, provided for compliance with RFC 4519, which stipulates that every `groupOfUniqueNames` object must contain at least one `uniqueMember`. `bjensen` is not actually used in this sample.

2. Prepare IDM, and start the server using the sample configuration:

```
cd /path/to/openidm/
./startup.sh -p samples/sync-with-ldap-groups
```

Run the Sample

You can run this sample using the command line or Admin UI:

+ *Use the Command Line*

1. Reconcile the group objects over the REST interface:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemLdapGroups_managedGroup&waitForCompletion=true"
{
  "id": "83f5b34b-0ddd-4c39-9349-de24816487ff-1198",
  "state": "SUCCESS"
}
```

The reconciliation operation returns a reconciliation run ID along with operation status, and creates managed group objects for each group that exists in DS.

2. To list the managed groups, run the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/group?_queryFilter=true"
{
  "result": [
    {
      "_id": "b6c4d7ce-2103-42c2-b5f2-74ca9309ad37",
      "_rev": "000000001298f6a6",
      "dn": "cn=Contractors,ou=Groups,dc=example,dc=com",
      "description": null,
      "uniqueMember": [],
      "name": "Contractors"
    },
    {
      "_id": "2326b9ee-6975-4c19-aa3c-d228afc4ff71",
      "_rev": "00000000dc6160c8",
      "dn": "cn=openidm2,ou=Groups,dc=example,dc=com",
      "description": null,
      "uniqueMember": [
        "uid=bjensen,ou=People,dc=example,dc=com"
      ],
      "name": "openidm2"
    },
    {
      "_id": "035f6444-bce3-4931-96b7-e10b2301fe74",
      "_rev": "000000004cab60c8",
      "dn": "cn=Employees,ou=Groups,dc=example,dc=com",
      "description": null,
      "uniqueMember": [],
      "name": "Employees"
    },
    {
      "_id": "65c8fb86-01e6-4fca-9237-e50c251f4575",
      "_rev": "0000000050c62938",
      "dn": "cn=Chat Users,ou=Groups,dc=example,dc=com",
      "description": null,
      "uniqueMember": [],
      "name": "Chat Users"
    },
    {
      "_id": "5c3e4965-16d7-4a8f-af73-3ab165b66cf9",
      "_rev": "000000004121fb7e",
      "dn": "cn=openidm,ou=Groups,dc=example,dc=com",
      "description": null,
      "uniqueMember": [
        "uid=jdoe,ou=People,dc=example,dc=com"
      ],
      "name": "openidm"
    }
  ],
  ...
}
```

```
}
```

+ Use the Admin UI

1. Log in to the Admin UI.

Warning

To protect your deployment in production, you must change the default administrative password. For more information, see ["Change the Administrator User Password"](#) in the *Security Guide*.

2. From the navigation bar, click Configure > Mappings.

The Mappings page displays three configured mappings:

- From the `ldap` server user accounts to the IDM repository (`managed/user`).
- From the IDM managed users back to the `ldap` accounts.
- From the `ldap` server group entries to the IDM `managed/group` entries.

3. Select the LDAP groups to managed groups mapping, and click Reconcile.

The reconciliation operation creates the two groups from the LDAP server in the IDM repository.

4. From the navigation bar, click Manage > Group.

IDM displays the five groups from the LDAP server (source) that were reconciled to the IDM repository (target).

Chapter 7

Synchronize LDAP Group Membership

This sample demonstrates synchronization between an LDAP directory and an IDM repository, with a focus on synchronizing LDAP group membership, that is, how the value of the `ldapGroups` property in a managed user object is mapped to the corresponding user object in LDAP.

The sample has been tested with ForgeRock Directory Services (DS) but should work with any LDAPv3-compliant server. The sample includes mappings from the LDAP server to the IDM repository, and from the IDM repository to the LDAP server. During the reconciliation, memberships are synchronized, in addition to user entries.

Prepare the Sample

1. Set up DS using `/path/to/openidm/samples/sync-with-ldap-group-membership/data/Example.ldif`.

The import file includes a number of LDAP groups, including the following:

```
dn: ou=Groups,dc=example,dc=com
ou: Groups
objectClass: organizationalUnit
objectClass: top

dn: cn=openidm,ou=Groups,dc=example,dc=com
uniqueMember: uid=jdoe,ou=People,dc=example,dc=com
cn: openidm
objectClass: groupOfUniqueNames
objectClass: top

dn: cn=openidm2,ou=Groups,dc=example,dc=com
uniqueMember: uid=bjensen,ou=People,dc=example,dc=com
cn: openidm2
objectClass: groupOfUniqueNames
objectClass: top
```

The users with DNs `uid=jdoe,ou=People,dc=example,dc=com` and `uid=bjensen,ou=People,dc=example,dc=com` are also imported with the `Example.ldif` file.

2. Prepare IDM, and start the server using the sample configuration:

```
cd /path/to/openidm/
./startup.sh -p samples/sync-with-ldap-group-membership
```

Run the Sample

You can work through the sample using the command line or Admin UI:

+ Use the Command Line

1. Reconcile the repository over the REST interface:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemLdapAccounts_managedUser&waitForCompletion=true"
{
  "_id": "6652c292-5309-40e5-b272-b74d67dd95c9-4",
  "state": "SUCCESS"
}
```

The reconciliation operation returns a reconciliation run ID and the status of the operation. Reconciliation creates user objects from LDAP in the IDM repository, assigning the new objects random unique IDs.

2. Retrieve the user IDs from the repository:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true&_fields=_id"
{
  "result": [
    {
      "_id": "1eaca03d-aef7-415a-99d9-bfd3f442ef51",
      "_rev": "0000000028e4e01e"
    },
    {
      "_id": "4e15c41e-6051-4150-8cde-b91c16397f25",
      "_rev": "00000000bb39e698"
    }
  ],
  ...
}
```

3. To retrieve individual user objects, include the ID in the URL. The following request retrieves the user object for John Doe:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/1eaca03d-aef7-415a-99d9-bfd3f442ef51"
{
  "_id": "1eaca03d-aef7-415a-99d9-bfd3f442ef51",
  "_rev": "0000000028e4e01e",
  "displayName": "John Doe",
  "description": "Created for OpenIDM",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "1-415-599-1100",
  "sn": "Doe",
  "userName": "jdoe",
  "ldapGroups": [
    "cn=openidm,ou=Groups,dc=example,dc=com"
  ],
  "accountStatus": "active",
  ...
}
```

Note

John Doe's user object contains an `ldapGroups` property, whose value shows his groups on the LDAP server:

```
"ldapGroups": ["cn=openidm,ou=Groups,dc=example,dc=com"]
```

4. Update John Doe's `ldapGroups` property, to change his membership from the `openidm` group to the `openidm2` group:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '[
{
  "operation": "replace",
  "field": "/ldapGroups",
  "value": ["cn=openidm2,ou=Groups,dc=example,dc=com"]
}
]' \
"http://localhost:8080/openidm/managed/user/1eaca03d-ae7f-415a-99d9-bfd3f442ef51?_action=patch"
{
  "displayName": "John Doe",
  "description": "Created for OpenIDM",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "1-415-599-1100",
  "sn": "Doe",
  "userName": "jdoe",
  "ldapGroups": [
    "cn=openidm2,ou=Groups,dc=example,dc=com"
  ],
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": [],
  "_rev": "00000000d0ebe04c",
  "_id": "1eaca03d-ae7f-415a-99d9-bfd3f442ef51"
}
```

This command changes John Doe's `ldapGroups` property in the IDM repository, from `"cn=openidm,ou=Groups,dc=example,dc=com"` to `"cn=openidm2,ou=Groups,dc=example,dc=com"`. Because of implicit synchronization, the change is propagated to the LDAP server. John Doe is removed from the first LDAP group and added to the second LDAP group in DS.

5. To verify the change, query John Doe's record on the LDAP server:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account/?_queryFilter=uid+eq+'jdoe'"
{
  "result": [
    {
      "_id": "0da50512-79bb-3461-bd04-241ee4c785bf",
      "employeeType": [],
      "objectClass": [
        "person",
        "organizationalPerson",
        "inetOrgPerson",
        "top"
      ],
      "cn": "John Doe",
```

```

    "uid": "jdoe",
    "ldapGroups": [
      "cn=openidm2,ou=Groups,dc=example,dc=com"
    ],
    "givenName": "John",
    "mail": "jdoe@example.com",
    "aliasList": [],
    "dn": "uid=jdoe,ou=People,dc=example,dc=com",
    "sn": "Doe",
    "description": "Created for OpenIDM",
    "telephoneNumber": "1-415-599-1100",
    "kbaInfo": []
  }
},
...
}

```

+ Use the Admin UI

1. Log in to the Admin UI.
2. From the navigation bar, click Configure > Mappings.

The Mappings page displays two configured mappings, one from the LDAP server to the IDM repository (**managed/user**) and one from the IDM repository to the LDAP server.

3. Select the LDAP to managed user mapping, and click Reconcile.

The reconciliation operation creates the two users from the LDAP server in the IDM repository.

4. To view the new users in the repository, from the navigation bar, click Manage > User.

IDM displays the two users.

5. From the Users List page, click the user **jdoe**.
6. Click the Linked Systems tab.

IDM displays John Doe's mapped external resource, **ldap/account**.

In the mapped resource, IDM displays John Doe's **ldapGroups**:

```
cn=openidm,ou=Groups,dc=example,dc=com
```

7. Update John Doe's **ldapGroups** property to change his membership from the **openidm** group to the **openidm2** group. You must perform this operation over the REST interface:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '[
{
  "operation": "replace",
  "field": "/ldapGroups",
  "value": ["cn=openidm2,ou=Groups,dc=example,dc=com"]
}
]' \
"http://localhost:8080/openidm/managed/user?_action=patch&_queryId=for-userName&uid=jdoe"
{
  "displayName": "John Doe",
  "description": "Created for OpenIDM",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "1-415-599-1100",
  "sn": "Doe",
  "userName": "jdoe",
  "ldapGroups": [
    "cn=openidm2,ou=Groups,dc=example,dc=com"
  ],
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": [],
  "_rev": "0000000050c62938",
  "_id": "8462fe0c-2ab2-459a-a25e-474474889c9e"
}
```

This command changes John Doe's `ldapGroups` property in the IDM repository, from `"cn=openidm,ou=Groups,dc=example,dc=com"` to `"cn=openidm2,ou=Groups,dc=example,dc=com"`. As a result of implicit synchronization, the change is propagated to the LDAP server. John Doe is removed from the first LDAP group and added to the second LDAP group in DS.

8. To verify the change: reload John Doe's user information, selecting Linked Systems, and examining the value of his `ldapGroups` property.
 - a. Reload John Doe's user information page.
 - b. Click the Linked Systems tab.

Observe the value of his `ldapGroups` property.

Chapter 8

Synchronize Data Between Two External Resources

This sample demonstrates synchronization between two external resources, routed through the IDM repository.

The resources are named **LDAP** and **AD** and represent two separate LDAP directories. In the sample both resources are simulated with simple CSV files.

The sample also demonstrates the (optional) configuration of an outbound email service. You can set up outbound email if you want to receive emailed reconciliation summaries.

Configure Email for the Sample

If you do not configure the email service, the functionality of the sample does not change. However, you might see the following message in the OSGi console when you run a reconciliation operation:

```
Email service not configured; report not generated.
```

To configure IDM to send a reconciliation summary by email, follow these steps:

1. Copy `external.email.json` from `samples/example-configurations/conf/` to the `conf/` directory of this sample:

```
cd /path/to/openidm/  
cp samples/example-configurations/conf/external.email.json samples/sync-two-external-resources/conf/
```

2. Edit `external.email.json` for outbound email.
3. In the `samples/sync-two-external-resources/script` directory, edit the `reconStats.js` script to reflect the correct email details.

Near the start of the file, locate the `var email` variable and update the values as required:

```
var email = {
  //UPDATE THESE VALUES
  from : "openidm@example.com",
  to : "youremail@example.com",
  cc : "idmadmin2@example.com,idmadmin3@example.com",
  subject : "Recon stats for " + global.mappingName,
  type : "text/html"
},
template,
...
```

Run the Sample

No external configuration is required for this sample. Before you start, prepare IDM as described in "Prepare IDM".

1. Start the server with the configuration of this sample:

```
cd /path/to/openidm/
./startup.sh -p samples/sync-two-external-resources
```

2. Examine the data files.

The CSV files that simulate the two LDAP resources are located in the `openidm/samples/sync-two-external-resources/data/` directory. Look at the contents of these files. Initially, the `csvConnectorLDAPData.csv` file contains one user and the `csvConnectorADData.csv` file contains no users.

3. Run a reconciliation operation to synchronize the contents of the simulated LDAP resource with the IDM repository.

You can run the reconciliation in the Admin UI (Configure > Mappings, click `systemLdapAccounts_managedUser`, then click Reconcile) or over the command-line as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemLdapAccounts_managedUser&waitForCompletion=true"
{
  "_id": "75e08ea9-411f-4c25-96b9-8e2396fb75aa-1062",
  "state": "SUCCESS"
}
```

The reconciliation creates a managed user in the IDM repository. You do not need to run a second reconciliation to synchronize the AD resource. Implicit synchronization propagates any change made to managed users in the repository to the simulated AD resource.

For more information about implicit synchronization, see "Types of Synchronization" in the *Synchronization Guide*.

- Review the contents of the simulated AD resource (`csvConnectorADData.csv`):

```
more /path/to/openidm/samples/sync-two-external-resources/data/csvConnectorADData.csv
"uid", "username", "firstname", "description", "email", "lastname"
"1", "Barbara", "bjensen@example.com", "Jensen"
```

This file should now contain the same user that was present in the `csvConnectorLDAPData.csv` file.

Alternatively, you can list users in the AD resource with the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ad/account?_queryId=query-all-ids"
{
  "result": [
    {
      "_id": "1",
      "name": "1"
    }
  ],
  ...
}
```

- Use the `_id` of the user to read the complete user record from the AD resource:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ad/account/1"
{
  "_id": "1",
  "firstname": "Barbara",
  "lastname": "Jensen",
  "email": [
    "bjensen@example.com"
  ],
  "name": "1"
}
```

- To verify that the sample is working, repeat the process.

Set up a second user in the `csvConnectorLDAPData.csv` file. The following example shows how that file might appear with a second user (`scarter`):

```
"uid", "username", "firstname", "description", "email", "lastname"
"1", "bjensen", "Barbara", "Created By CSV", "bjensen@example.com", "Jensen"
"2", "scarter", "Steve", "Created By CSV", "scarter@example.com", "Carter"
```

- Rerun the reconciliation and query REST commands shown previously.

The reconciliation operation creates the new user from the simulated LDAP resource in the IDM repository. An implicit synchronization operation then creates that user in the AD resource.

8. If you configured the reconciliation email summary at the beginning of this sample, you should have received an email that lists the details of the reconciliation operations.

Chapter 9

Asynchronous Reconciliation Using a Workflow

This sample demonstrates asynchronous reconciliation using workflows.

The data for this sample is in the file `samples/sync-asynchronous/data/csvConnectorData.csv`. That file contains two users, as follows:

```
"description", "uid", "username", "firstname", "lastname", "email", "mobile..."
"Created ...", "bjensen", "bjensen@example.com", "Barbara", "Jensen", "bjensen@example.com", 1234..."
"Created ...", "scarter", "scarter@example.com", "Steven", "Carter", "scarter@example.com", 1234..."
```

During the sample, you will reconcile the users in the CSV file with the managed user repository. Instead of creating each user immediately, the reconciliation operation generates an approval request for each ABSENT user (users who are not found in the repository). The configuration for this action is defined in the `conf/sync.json` file, which specifies that an **ABSENT** condition should launch the `managedUserApproval` workflow:

```
...
{
  "situation" : "ABSENT",
  "action" : {
    "workflowName" : "managedUserApproval",
    "type" : "text/javascript",
    "file" : "workflow/triggerWorkflowFromSync.js"
  }
},
...
```

When each request is approved by an administrator, an asynchronous reconciliation operation is launched, that ultimately creates the users in the repository.

Run the Sample

Before you start, prepare IDM as described in "Prepare IDM".

Important

Workflows are not supported with a DS repository. Before you test this sample, install a JDBC repository.

1. Edit the `/path/to/openidm/samples/sync-asynchronous/conf/datasource.jdbc-default.json` file with the details of your JDBC repository. For more information, see "Select a Repository" in the *Installation Guide*.

2. Start IDM with the configuration for this sample:

```
/path/to/openidm/startup.sh -p samples/sync-asynchronous
```

3. The sample is configured to assign new workflow tasks to an admin account named `async.admin`. Create this account before you begin:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "userName": "async.admin",
  "givenName": "async",
  "sn": "admin",
  "password": "Passw0rd",
  "displayName": "async admin",
  "mail": "async.admin@example.com",
  "authzRoles": [
    "internal/role/openidm-admin",
    "internal/role/openidm-authorized"
  ],
  "_id": "asynccadmin"
}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "_id": "asynccadmin",
  "_rev": "00000000e8f502db",
  "userName": "async.admin",
  "givenName": "async",
  "sn": "admin",
  "displayName": "async admin",
  "mail": "async.admin@example.com",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

4. Run reconciliation over the REST interface:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?_action=recon&mapping=systemCsvfileAccounts_managedUser"
{
  "_id": "98d7f3c5-684e-4ef0-b4f9-f2e816a339cf-32",
  "state": "ACTIVE"
}
```

The reconciliation operation returns a reconciliation run ID, and the status of the operation.

This reconciliation launches a workflow that generates an approval process for each ABSENT user. The approval processes must be approved by an administrator before the workflow can continue.

5. Review the approval tasks launched by the reconciliation.

- To review the tasks in the Admin UI, log in to the Admin UI at <https://localhost:8443/admin/> using an administrator account (either `openidm-admin` or `async.admin` will work) and select Manage > Tasks.

You should see two task instances launched by the Managed User Approval Workflow.

- To view the approval tasks over REST run the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/workflow/taskinstance?_queryFilter=true&_fields=_id,processDefinitionId"
```

The request returns two task instances, each with a process ID (`_id`) and a process definition ID.

```
{
  "result": [
    {
      "_id": "38",
      "processDefinitionId": "managedUserApproval:1:5"
    },
    {
      "_id": "39",
      "processDefinitionId": "managedUserApproval:1:5"
    }
  ],
  ...
}
```

6. Complete each approval task.

- To complete the approval tasks using the UI, log in to the End User UI at <https://localhost:8443/#/login> as user `async.admin` with password `Passw0rd`.

You should see two *Evaluate request* tasks under My Tasks on the Dashboard.

For each task, select Edit and then select Approve to add the noted users.

- To approve the requests over REST, set `requestApproved` to `true` for each task instance, and use the `complete` action. Specify the `_id` of each task in the URL.

For example, to approve the first request:

```
curl \
--header "X-OpenIDM-Username: async.admin" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{"requestApproved": "true"}' \
"http://localhost:8080/openidm/workflow/taskinstance/38?_action=complete"
{
  "Task action performed": "complete"
}
```

Repeat this command for each task ID.

7. When the requests have been approved, select Manage > User in the Admin UI to see the new users in the repository, or query the managed users over REST as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true&_fields=_id"
{
  "result": [
    {
      "_id": "asyncadmin",
      "_rev": "00000000e8f502db"
    }, {
      "_id": "scarter",
      "_rev": "000000007e120780"
    }, {
      "_id": "bjensen",
      "_rev": "00000000d9390751"
    }
  ],
  ...
}
```

Chapter 10

LiveSync With an LDAP Server

This sample resembles the sample described in "[Synchronize Data Between Two External Resources](#)". However, this sample demonstrates *liveSync* from one external LDAP resource to another. LiveSync is the mechanism by which changes are pushed from an external resource to IDM and then, optionally, to another external resource. For more information see "Types of Synchronization" in the *Synchronization Guide*.

The sample assumes a scenario where changes in an Active Directory server are synchronized, using LiveSync, with a ForgeRock Directory Services (DS) server.

The sample provides a configuration for two scenarios, depending on whether you are using a live Active Directory (AD) service, or whether you are simulating the AD service with a DS server. Each scenario is associated with a file in the `livesync-with-ad/alternatives` directory. Depending on your scenario, copy the corresponding file to the `livesync-with-ad/conf` directory:

Live AD Instance

If you have an AD instance to test with, use that AD instance as the first LDAP resource. For the second LDAP resource, configure DS as described in "[Set Up the LDAP Resources](#)". The data for the DS instance is contained in the file `samples/livesync-with-ad/data/Example.ldif`.

For the connection to Active Directory, copy the `provisioner.openicf-realad.json` file to the `conf/` directory and rename it `provisioner.openicf-ad.json`.

Because this sample demonstrates synchronization *from* the AD server *to* DS, data on the AD server is not changed.

Simulated AD Instance

If you are simulating the AD instance with a DS server, copy the `provisioner.openicf-fakead.json` file to the `conf/` subdirectory and rename it `provisioner.openicf-ad.json`.

This sample simulates an AD server on the same instance of DS, using a different base DN (`dc=fakead,dc=com`). You can also simulate the AD server with a separate DS instance, running on the same host, as long as the two instances communicate on different ports. The data for the simulated AD instance is contained in the file `samples/livesync-with-ad/data/AD.ldif`. The data for the DS instance is contained in the file `samples/livesync-with-ad/data/Example.ldif`.

Set Up the LDAP Resources

Whether you use a simulated Active Directory server, or a live Active Directory server, you must still set up a DS instance as the second LDAP resource.

Set up DS using `/path/to/openidm/samples/livesync-with-ad/data/Example.ldif`, and do one of the following:

+ *Configure the Connection to a Live AD Instance*

To configure the connection to a live AD instance, open the connector configuration file (`provisioner.openicf-ad.json`) in a text editor. Update the file as required to reflect your AD instance. At a minimum, check and update the following parameters:

host

The hostname or IP address of the AD server.

port

The LDAP port; 389 by default.

ssl

Whether the connection to the AD instance is secured over SSL; false by default.

principal

The full DN of the account that is used to bind to the server, for example, "CN=Administrator,CN=Users,DC=example,DC=com".

credentials

If a password is used, replace null with that password. When IDM starts, it encrypts that password in the `provisioner.openicf-ad.conf` file.

baseContexts

A list of DNs for account containers, for example, "CN=Users,DC=Example,DC=com".

baseContextsToSynchronize

Set to the same value as `baseContexts`.

accountSearchFilter

The LDAP search filter to locate accounts; only user accounts by default.

accountSynchronizationFilter

The LDAP search filter to synchronize user accounts; only user accounts by default.

If you do not want to filter out computer and disabled user accounts, set the `accountSearchFilter` and `accountSynchronizationFilter` to `null`.

+ Configure the Connection to a Simulated AD Instance

If you do not have a testable instance of AD available, you can simulate an AD instance in a separate suffix on the existing DS instance. The `data/AD.ldif` file includes LDIF data for a simulated AD instance.

Configure an Existing DS Instance as a Simulated AD Instance

1. If you have not already done so, copy the `samples/livesync-with-ad/alternatives/provisioner.openicf-fakead.json` to the `conf` subdirectory and rename it `provisioner.openicf-ad.json`.

This file sets up the connection to the DS server, targeting the suffix (`dc=fakead,dc=com`) that is simulating an AD server.

2. Load the simulated data into that suffix:

```
/path/to/openssl/bin/openssl \
--port 1636 \
--useSSL \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--hostname localhost \
--bindDN uid=admin \
--bindPassword password \
--filename /path/to/openssl/samples/livesync-with-ad/data/AD.ldif
# ADD operation successful for DN dc=fakead,dc=com

# ADD operation successful for DN ou=People,dc=fakead,dc=com

# ADD operation successful for DN uid=bobf,ou=People,dc=fakead,dc=com

# ADD operation successful for DN uid=stony,ou=People,dc=fakead,dc=com
```

Run the Sample

When both DS and a real or simulated AD server are configured, prepare IDM as described in "Prepare IDM". Then start IDM with the configuration for this sample:

```
cd /path/to/openssl/
./startup.sh -p samples/livesync-with-ad
```

The following sections show how to synchronize the two external LDAP data stores by running a reconciliation operation, how to configure scheduled liveSync.

Reconcile the Two LDAP Data Stores

Review the entries in the DS server (imported from the `Example.ldif` file). When you run reconciliation, any entries that share the same `uid` with the AD data store will be updated with the contents from AD.

If you have set up the simulated AD data store as described in [Configure the Connection to a Simulated AD Instance](#), compare the entries in the `AD.ldif` and `Example.ldif` files. Note that each file has two different users—after importing the data, the AD instance has users `bobf` and `stony`, and the DS instance has users `jdoe` and `bjensen`.

Run reconciliation over the REST interface:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemAdAccounts_managedUser&waitForCompletion=true"
```

The reconciliation operation returns a reconciliation run ID, and the status of the operation.

```
{
  "state": "SUCCESS",
  "_id": "985ee939-fbe1-4607-a757-00b404b4ef77"
}
```

The reconciliation operation synchronizes the data in the AD server with the IDM repository (managed/user). *Implicit synchronization* then pushes those changes out to the DS server. For more information about implicit synchronization, see "Types of Synchronization" in the *Synchronization Guide*.

List all the users in the DS server:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account?_queryFilter=true&_fields=_id,dn"
```

Your DS server should now contain four users:

```
{
  "result": [
    {
      "_id": "0da50512-79bb-3461-bd04-241ee4c785bf",
      "dn": "uid=jdoe,ou=People,dc=example,dc=com"
    },
    {
      "_id": "887732e8-3db2-31bb-b329-20cd6fcecc05",
      "dn": "uid=bjensen,ou=People,dc=example,dc=com"
    },
    {
      "_id": "fc4feff0-11ae-430f-858d-338b1b05d66a",
      "dn": "uid=bobf,ou=People,dc=example,dc=com"
    },
    {
      "_id": "ba07d4f4-0e2b-4c53-aecb-4b234e1fec1c",
      "dn": "uid=stony,ou=People,dc=example,dc=com"
    }
  ],
  ...
}
```

Note that the two users from the AD server have been added to the DS server.

Configure LiveSync

LiveSync pushes changes made in an external system to the IDM repository. You can launch a liveSync operation over REST, or configure a schedule to poll for changes. This sample includes a liveSync schedule ([conf/schedule-activeSynchroniser_systemAdAccount.json](#)) that is disabled by default. When the schedule is enabled, a liveSync operation is launched every 15 seconds.

To activate liveSync, change the value of the **enabled** property in the schedule configuration from **false** to **true**:

```
{
  "enabled" : true,
  "type" : "simple",
  "repeatInterval" : 15000,
  "persisted" : true,
  "concurrentExecution" : false,
  "invokeService" : "provisioner",
  "invokeContext" : {
    "action" : "liveSync",
    "source" : "system/ad/account"
  },
  "invokeLogLevel" : "debug"
}
```

Test the liveSync as follows:

1. Create an LDIF file with a new user entry (**uid=bsmith**) that will be added to the AD directory.

You can use the following LDIF file (**bsmith.ldif**) as an example. This sample file assumes the simulated AD instance. Adjust the DN if you are using a live AD instance:

```
dn: uid=bsmith,ou=People,dc=fakead,dc=com
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: top
givenName: Barry
description: Created to see liveSync work
uid: bsmith
cn: Barry
sn: Smith
mail: bsmith@example.com
telephoneNumber: 1-415-523-0772
userPassword: 5up35tr0ng
```

2. Use the **ldapmodify** command to add the new user to the AD directory:

```
/path/to/openssl/bin/ldapmodify \
--port 1636 \
--useSSL \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--hostname localhost \
--bindDN uid=admin \
--bindPassword password \
--filename /path/to/bsmith.ldif
# ADD operation successful for DN uid=bsmith,ou=People,dc=fakead,dc=com
```

3. Within 15 seconds, liveSync should create the user in the IDM repository.

Test that the liveSync has worked by viewing the new user in IDM.

The easiest way to do this, is through the End User UI. You should be able to log in to the End User UI (at <http://localhost:8080>) as user **bsmith**, with password **5up35tr0ng**. If you can log in to the UI as this new user, liveSync has synchronized the user from the AD directory to the managed/user repository.

4. Implicit synchronization pushes this change out to the DS server. To test this synchronization operation, search the DS baseDN for the new user entry.

```
/path/to/openssl/bin/ldapsrch \
--port 1636 \
--useSSL \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--hostname localhost \
--bindDN uid=admin \
--bindPassword password \
--baseDN ou=people,dc=example,dc=com \
"(uid=bsmith)"
```

Chapter 11

Synchronize Accounts With the Google Apps Connector

The Google Apps Connector lets you interact with Google's web applications.

This sample shows how to create users and groups on an external Google system, and how to synchronize those accounts with the IDM managed user repository. The sample requires a Google Apps account.

Prepare the Sample

To set up IDM to connect to your Google Apps account, you must have a Google Apps project that authorizes consent for IDM.

1. Log in to the [Google Apps Developers Console](#) and update your existing project, or create a new project for IDM.
2. Enable the following APIs for your IDM project:
 - Admin SDK API
 - Enterprise License Manager API
3. Set up an OAuth2 Client.

The Google Apps connector uses OAuth2 to authorize the connection to the Google service:

- a. In the Google Apps Developers Console, select Credentials > Create Credentials > OAuth client ID.
- b. Click Configure Consent Screen, select Internal, and click Create.
- c. On the OAuth consent screen, enter an Application name; for example, **IDM**, and click Save.

This is the name that will be shown for all applications registered in this project.

- d. Select Credentials > Create Credentials > OAuth client ID > Web application, then complete these fields:

- Authorized JavaScript origins: the URI at which your clients will access your application. The default URI is `https://localhost:8443`.

Note

The URI that you enter here must be the same URI at which you access IDM. If you enter `https://localhost:8443` here, but use `http://localhost:8080` to access IDM, the sample will not work.

- Authorized redirect URIs: the OAuth redirect URI, `https://localhost:8443/admin/oauth.html` by default.

Click Create.

- e. On the OAuth client created popup, make a note of your Client ID and Client Secret.

Copy and paste these values into a text file, as you will need them when you configure the Google Apps connector.

4. Add IDM to the Trusted Apps list.

- a. Log in to the Google Admin Console.
- b. From the top left menu, select Security > API Controls.
- c. Select MANAGE THIRD-PARTY APP ACCESS, then click Change Access, and change the IDM app settings to Trusted.

Configure the Google Apps Connector

This procedure uses the Admin UI to configure the Google Apps connector.

1. Start IDM with the Google Apps sample configuration:

```
/path/to/openidm/startup.sh -p samples/sync-with-google
Executing ./startup.sh...
Using OPENIDM_HOME: /path/to/openidm
Using PROJECT_HOME: /path/to/openidm/samples/sync-with-google/
Using OPENIDM_OPTS: -Xmx1024m -Xms1024m
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
Using boot properties at /path/to/openidm/resolver/boot.properties
-> OpenIDM ready
```









2. Log in to the Admin UI at the URL `https://localhost:8443/admin` as the default administrative user (`openidm-admin`) with password `openidm-admin`.

This URL reflects the host on which IDM is installed, and must be the same as the **Authorized JavaScript origin** URI that you set in your Google app.

3. Select Configure > Connectors and click on the Google Apps connector.
4. On the Details tab, set Enabled to True.
5. Enter the OAuth2 Client ID and Client Secret that you obtained in the previous section.
6. Click Save Connector Changes.
7. You are redirected to Google's Login page.

When you have logged in, Google requests that you allow access from your project, in this case, IDM.

▼ OpenIDM would like to:

	View and manage the provisioning of users on your domain	
	View and manage the provisioning of groups on your domain	
	View and manage Google Apps licenses for your domain	
	View and manage organization units on your domain	

By clicking Allow, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other [Account Permissions](#) at any time.

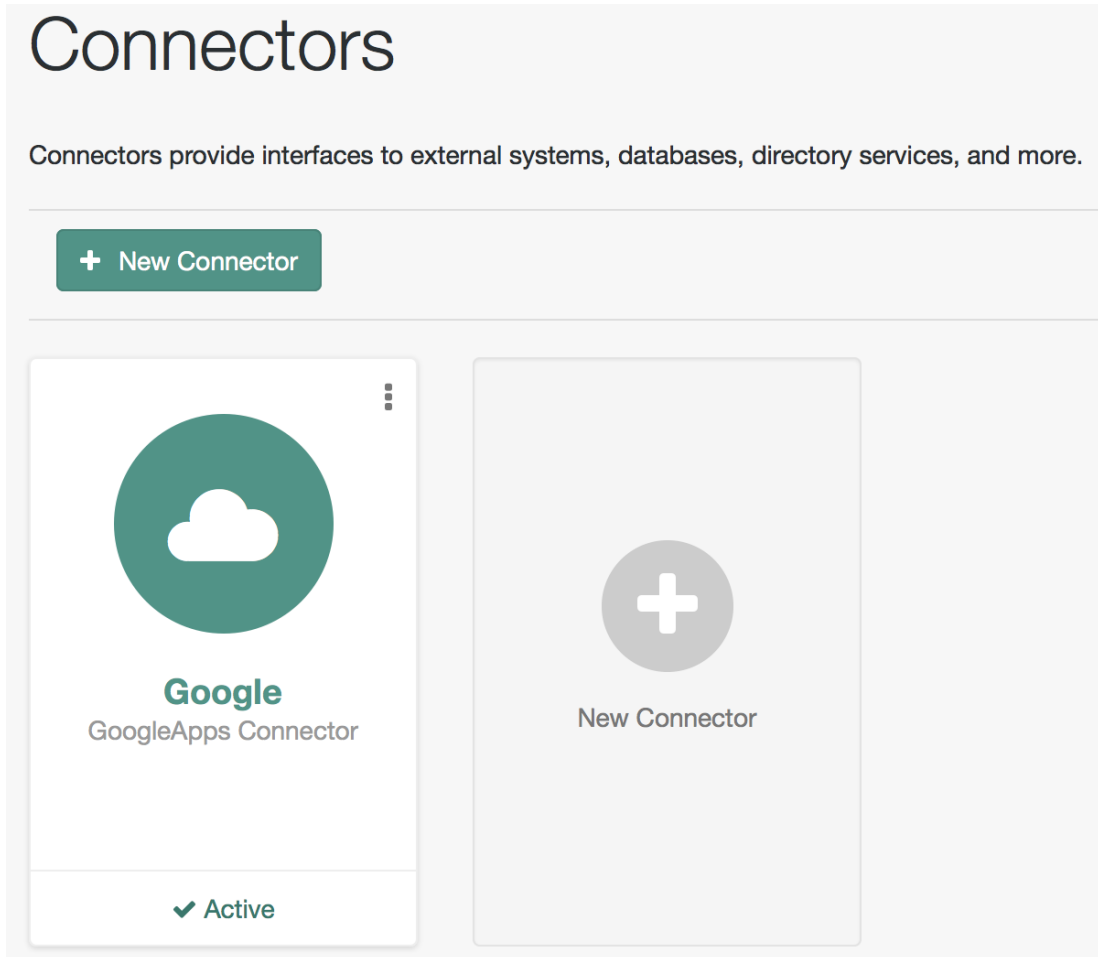
Deny

Allow

Click Allow.

If you click Deny here, you will need to return to the Connector Configuration > Details tab in the Admin UI and save your changes again.

When you allow access, you are redirected to the Connectors page in the Admin UI, where the Google Apps Connector should now be Active.



Run the Sample

This procedure uses create, read, update, and delete (CRUD) operations on the Google resource, to verify that the connector is working as expected. The procedure uses a combination of REST

commands, to manage objects on the Google system, and the Admin UI, to reconcile users from the Google system to the manage user repository.

The sample configuration has one mapping *from* the Google system to the managed user repository.

All of the commands shown here assume that your domain is `example.com`. Adjust the examples to manage your domain.

1. Create a user entry on your Google resource, over REST.

When you create resources for Google, note that the equals (=) character cannot be used in any attribute value.

The following command creates an entry for user `Sam Carter`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "__NAME__": "samcarter@example.com",
  "__PASSWORD__": "password",
  "givenName": "Sam",
  "familyName": "Carter",
  "agreedToTerms": true,
  "changePasswordAtNextLogin": false
}' \
"http://localhost:8080/openidm/system/google/__ACCOUNT__?_action=create"
{
  "_id": "103567435255251233551",
  "_rev": "\"iwppoDgSq9BJw-XzORg0bILYPVc/LWHPMXXG8M0cjQAPITM95Y636cM\"",
  "orgUnitPath": "/",
  "isAdmin": false,
  "fullName": "Sam Carter",
  "customerId": "C02rsqddz",
  "relations": null,
  "nonEditableAliases": null,
  "suspensionReason": null,
  "includeInGlobalAddressList": true,
  "givenName": "Sam",
  "addresses": null,
  "isDelegatedAdmin": false,
  "changePasswordAtNextLogin": false,
  "isMailboxSetup": true,
  "__NAME__": "samcarter@example.com",
  "agreedToTerms": true,
  "externalIds": null,
  "ipWhitelisted": false,
  "aliases": null,
  "lastLoginTime": [
    "1970-01-01T00:00:00.000Z"
  ],
  "organizations": null,
  "suspended": false,
  "deletionTime": null,
```

```
{
  "familyName": "Carter",
  "ims": null,
  "creationTime": [
    "2016-02-02T12:52:30.000Z"
  ],
  "thumbnailPhotoUrl": null,
  "emails": [
    {
      "address": "samcarter@example.com",
      "primary": true
    }
  ],
  "phones": null
}
```

Note the ID of the new user (**103567435255251233551** in this example). You will need this ID for the update commands in this section.

2. Reconcile the Google resource with the managed user repository.

This step should create the new user, Sam Carter (and any other users in your Google resource) in the managed user repository.

To run reconciliation follow these steps:

- In the Admin UI, select Configure > Mappings.
 - Click on the sourceGoogle__ACCOUNT__managedUser mapping, and click Reconcile.
 - Select Manage > User and verify that the user Sam Carter has been created in the repository.
- ## 3. Update Sam Carter's phone number in your Google resource by sending a PUT request with the updated data, and specifying the user **_id** in the request:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PUT \
--header "If-Match: *" \
--data '{
  "__NAME__": "samcarter@example.com",
  "__PASSWORD__": "password",
  "givenName": "Sam",
  "familyName": "Carter",
  "agreedToTerms": true,
  "changePasswordAtNextLogin": false,
  "phones": [
    {
      "value": "1234567890",
      "type": "home"
    },
    {
      "value": "0987654321",

```

```

    "type": "work"
  }
} \
"http://localhost:8080/openidm/system/google/___ACCOUNT___/103567435255251233551"
{
  "_id": "103567435255251233551",
  "_rev": "\"iwpzoDgSq9BJw-XzORg0bILYPVc/vfSJgHt-STUUt04lM_4ES09izR4\"",
  ...
  "emails": [
    {
      "address": "samcarter@example.com",
      "primary": true
    }
  ],
  "phones": [
    {
      "value": "1234567890",
      "type": "home"
    },
    {
      "value": "0987654321",
      "type": "work"
    }
  ]
}
}

```

4. Read Sam Carter's entry from your Google resource by including his `_id` in the URL:

```

curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/google/___ACCOUNT___/103567435255251233551"
{
  "_id": "103567435255251233551",
  "___NAME___": "samcarter@example.com",
  ...
  "phones": [
    {
      "value": "1234567890",
      "type": "home"
    },
    {
      "value": "0987654321",
      "type": "work"
    }
  ]
}
}

```

5. Create a group entry on your Google resource:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "__NAME__": "testGroup@example.com",
  "__DESCRIPTION__": "Group used for google-connector sample.",
  "name": "TestGroup"
}' \
"http://localhost:8080/openidm/system/google/__GROUP__?_action=create"

{
  "_id": "00meukdy40gpg98",
  "_rev": "\"iwpzoDgSq9BJw-XzORg0bILYPVc/LLhHx2pLMJPKeY1-h6eX_OVDi4c\"",
  "adminCreated": true,
  "__NAME__": "testgroup@example.com",
  "aliases": null,
  "nonEditableAliases": null,
  "__DESCRIPTION__": "Group used for google-connector sample.",
  "name": "TestGroup",
  "directMembersCount": 0
}
```

6. Add Sam Carter to the test group you have just created. Include the **Member** endpoint, and Sam Carter's **_id** in the URL. Specify the **_id** of the group you created as the value of the **groupKey** in the JSON payload:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
-H 'If-Match: "iwpzoDgSq9BJw-XzORg0bILYPVc/LLhHx2pLMJPKeY1-h6eX_OVDi4c"' \
--request PUT \
--data '{
  "__MEMBERS__": [
    {
      "role": "MEMBER",
      "email": "samcarter@example.com"
    }
  ]
}' \
"http://localhost:8080/openidm/system/google/__GROUP__/00meukdy40gpg98"

{
  "_id": "00meukdy40gpg98/samcarter@example.com",
  "_rev": "iwpzoDgSq9BJw-XzORg0bILYPVc/CPNpkRnowkGWRvNQvUK9ev6gQ90",
  "__NAME__": "00meukdy40gpg98/samcarter@example.com",
  "role": "MEMBER",
  "email": "samcarter@example.com",
  "type": "USER",
  "groupKey": "103567435255251233551"
}
```

7. Read the group entry by specifying the group **_id** in the request URL. Notice that the group has one member (**"directMembersCount": 1**):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/google/___GROUP___/00meukdy40gpg98"

{
  "_id": "00meukdy40gpg98",
  "_rev": "iwpzoDgSq9BJw-Xz0Rg0bILYPVc/chUdq5m5_cycV2G4sdl7ZKAF75A",
  "adminCreated": true,
  "___NAME___": "testgroup@example.com",
  "aliases": null,
  "nonEditableAliases": [
    "testGroup@example.test-google-a.com"
  ],
  "___DESCRIPTION___": "Group used for google-connector sample.",
  "name": "TestGroup",
  "directMembersCount": 1
}
```

8. Delete the group entry:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/system/google/___GROUP___/00meukdy40gpg98"

{
  "_id": "00meukdy40gpg98",
  "_rev": "iwpzoDgSq9BJw-Xz0Rg0bILYPVc/chUdq5m5_cycV2G4sdl7ZKAF75A",
  "adminCreated": true,
  "___NAME___": "testgroup@example.com",
  "aliases": null,
  "nonEditableAliases": [
    "testGroup@example.com.test-google-a.com"
  ],
  "___DESCRIPTION___": "Group used for google-connector sample.",
  "name": "TestGroup",
  "directMembersCount": 1
}
```

The delete request returns the complete group object.

9. Delete Sam Carter, to return your Google resource to its original state:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/system/google/___ACCOUNT___/103567435255251233551"

{
  "_id": "103567435255251233551",
  "_rev": "iwpzoDgSq9BJw-Xz0Rg0bILYPVc/ah6xBLujMAHieSWsisPa1CV6T3Q",
  "orgUnitPath": "/",
  "isAdmin": false,
}
```

```

"fullName": "Sam Carter",
"customerId": "C02rsqddz",
"relations": null,
"nonEditableAliases": [
  "samcarter@example.com.test-google-a.com"
],
"suspensionReason": null,
"includeInGlobalAddressList": true,
"givenName": "Sam",
"addresses": null,
"isDelegatedAdmin": false,
"changePasswordAtNextLogin": false,
"isMailboxSetup": true,
"__NAME__": "samcarter@example.com",
"agreedToTerms": true,
"externalIds": null,
"ipWhitelisted": false,
"aliases": null,
"lastLoginTime": [
  "1970-01-01T00:00:00.000Z"
],
"organizations": null,
"suspended": false,
"deletionTime": null,
"familyName": "Carter",
"ims": null,
"creationTime": [
  "2016-02-02T12:52:30.000Z"
],
"thumbnailPhotoUrl": null,
"emails": [
  {
    "address": "samcarter@example.com",
    "primary": true
  }
],
"phones": [
  {
    "value": "1234567890",
    "type": "home"
  },
  {
    "value": "0987654321",
    "type": "work"
  }
]
}

```

In this sample, you used the Google Apps connector to add and delete user and group objects in your Google application, and to reconcile users from your Google application to the managed user repository. You can expand on this sample by customizing the connector configuration to provide additional synchronization functionality between IDM and your Google applications. For more information on configuring connectors, see [Overview](#) in the *Connectors Guide*.

Chapter 12

Synchronize Users Between Salesforce and IDM

The Salesforce connector enables provisioning, reconciliation, and synchronization between Salesforce and IDM.

This sample shows how to synchronize Salesforce user accounts and managed users in the IDM repository. You can use either the Admin UI, or the command line to run this sample. Both methods are outlined in the sections that follow.

Prepare the Sample

1. Configure your Salesforce organization.

To test this sample you must have an existing Salesforce organization, a Salesforce developer account, and a Connected App with OAuth enabled. For instructions on setting up a Connected App, see the corresponding [Salesforce documentation](#). When you have set up the Connected App, locate the *Consumer Key* and *Consumer Secret*. You will need these details to configure the connector.

When you set up your Connected App, make sure that you include the following scopes, even if you plan to use the "Full access (full)" scope:

- Access and manage your data (api).
- Access your basic information (id, profile, email, address, phone).
- Perform requests on your behalf at any time (refresh_token, offline_access).

2. Prepare IDM as described in "Prepare IDM", then start the server with the configuration for the Salesforce sample:

```
/path/to/openidm/startup.sh -p samples/sync-with-salesforce
```

Run the Sample

You can run the sample using the Admin UI, or over the command line. Using the Admin UI is recommended because the command-line example is significantly more complex for this sample:

+ Use the Admin UI

1. Log in to the Admin UI at the URL <https://localhost:8443/admin> as the default administrative user (`openidm-admin`) with password `openidm-admin`.

Warning

To protect your deployment in production, change the default administrative password, as described in "Change the Administrator User Password" in the *Security Guide*.

2. Enable the Salesforce connector by completing the authentication details as follows. You will need the Consumer Key and Consumer Secret that you obtained from your Connected App configuration.

- Select the Salesforce connector and select Enable.
- Under Base Connector Details select Production, Sandbox or Custom to set your Login URL.

The Login URL is the OAuth endpoint that will be used to make the OAuth authentication request to Salesforce.

The default endpoint for a production system is <https://login.salesforce.com/services/oauth2/token>. The default endpoint for a sandbox (test) system is <https://test.salesforce.com/services/oauth2/token>.

Note

When you create your connected app, you are instructed to wait 2-10 minutes for the settings to propagate across all the Salesforce data centers. If you are using a Salesforce test tenant, such as <https://eu26.lightning.force.com>, you can specify a custom URL here and enter the FQDN of the test tenant. This will enable you to test the connector without waiting for the new app settings to be propagated.

- Enter your Consumer Key and Consumer Secret and select Save to update the connector configuration.
- The connector now attempts to access your Salesforce organization.

Enter your Salesforce login credentials.

On the permission request screen click Allow, to enable IDM to access your Salesforce Connected App.

3. To test reconciliation, select Configure > Mappings.

There are two configured mappings, one from Salesforce to the IDM repository (`managed/user`) and one from the repository to Salesforce.

4. Select Reconcile on the first mapping.

The reconciliation operation creates the users that were present in your Salesforce organization in the IDM repository.

5. Retrieve the users in the repository by selecting Manage > User.

The repository should now contain all the users from your Salesforce organization.

6. To test the second mapping (from IDM to Salesforce), update any user in the repository.

By default, *implicit synchronization* is enabled for mappings from the `managed/user` repository to any external resource. This means that when you update a managed object, any mappings defined in the `sync.json` file that have the managed object as the source are automatically run to update the target system. For more information, see "*Mapping Data Between Resources*" in the *Synchronization Guide*.

To test that the implicit synchronization has been successful, check the updated user record in Salesforce.

+ Use the Command Line

This section breaks the sample into two tasks—configuring the connector, and then testing the configuration by running reconciliation operations between the two systems:

Set Up the Salesforce Connector

1. Retrieve all the required configuration properties, as described in "To Configure the Salesforce Connector With a Configuration File" in the *Connectors Guide*.
2. Edit the `configurationProperties` object in the Salesforce connector configuration file (`openidm/samples/sync-with-salesforce/conf/provisioner.openicf-salesforce.json`) to include your Salesforce login URL, Consumer Key and Consumer Secret, refresh token, and instance URL.

Set the `enabled` property to `true` to enable the connector.

The relevant excerpts of the `provisioner.openicf-salesforce.json` file are as follows:

```
{
  "name" : "salesforce",
  "enabled" : true,
  ...
  "configurationProperties" : {
    "connectTimeout" : 120000,
    "loginUrl" : "https://login.salesforce.com/services/oauth2/token",
    "idleCheckInterval" : 10000,
    "refreshToken" : "5Aep861KIwKdekr90I4iHdtDgWwRoG70_6uHrgJ.yVtMS0UaGxRqE6WFM...",
    "clientSecret" : "4850xxxxxxxxxxxx425",
    "clientId" : "3MVG98dostKihXN7Is8Q0g5qlxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxP...",
    "instanceUrl" : "https://example-com.cs1.my.salesforce.com",
    "version" : 44
  }
  ...
}
```

3. Check that your connector configuration is correct by testing the status of the connector, over REST.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=test"
{
  "name": "salesforce",
  "enabled": true,
  "config": "config/provisioner.openicf/salesforce",
  "connectorRef": {
    "bundleVersion": "1.5.20.11",
    "bundleName": "org.forgerock.openicf.connectors.salesforce-connector",
    "connectorName": "org.forgerock.openicf.connectors.salesforce.SalesforceConnector"
  },
  "displayName": "Salesforce Connector",
  "objectTypes": [
    "ALL",
    "User"
  ],
  "ok": true
}
```

Run Reconciliation

The mapping configuration file ([sync.json](#)) for this sample includes two mappings, [systemSalesforceUser_managedUser](#), which synchronizes users from the Salesforce with the IDM repository, and [managedUser_systemSalesforceUser](#), which synchronizes changes from the repository to Salesforce.

1. Reconcile the repository over the REST interface by running the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemSalesforceUser_managedUser&waitForCompletion=true"
{
  "state": "SUCCESS",
  "_id": "8a6281ef-6faf-43dd-af5c-3a842b38c468"
}
```

The reconciliation operation returns a reconciliation run ID and the status of the operation. Reconciliation creates user objects from Salesforce in the IDM repository, assigning the new objects random unique IDs.

2. Retrieve the managed users in the repository:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true&_fields=_id"
{
  "result": [
    {
      "_id": "180c6686-b098-460a-a246-4e03fa0b8eb2",
      "_rev": "00000000cfe1fccf"
    },
    {
      "_id": "d0c25a0c-f7e6-4249-9c81-e546728f5bdd",
      "_rev": "000000000828e760"
    },
    {
      "_id": "25181ab3-0d40-4f80-96d6-d620eef7b6da",
      "_rev": "0000000038b6e342"
    }
  ],
  "resultCount": 3,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

The output shows that the users in the Salesforce data store have been created in the repository.

Chapter 13

Synchronize Kerberos User Principals

This sample demonstrates how to manage Kerberos user principals and how to reconcile user principals with IDM managed user objects.

The connector configuration (`/path/to/openidm/samples/sync-with-kerberos/conf/provisioner.openidm-kerberos.json`) assumes that IDM is running on a host that is separate from the Kerberos host.

This sample assumes that the default realm is `EXAMPLE.COM` and that there is an existing user principal `openidm/admin`. Adjust the sample to match your Kerberos realm and principals.

Configure the Kerberos Connector

Before you run this sample, edit the connector configuration file to match your Kerberos environment. Specifically, set the correct values for the following properties:

host

The host name or IP address of the machine on which Kerberos is running.

port

The SSH port on that machine.

Default: `22` (the default SSH port)

user

The username of the account that is used to connect to the SSH server.

password

The password of the account that is used to connect to the SSH server.

prompt

A string that represents the remote SSH session prompt. This must be the exact prompt string, in the format `username@target:`, for example `root@localhost:~$`. The easiest way to obtain this string is to `ssh` into the machine and copy paste the prompt.

customConfiguration

The details of the admin user principal and the default realm.

This example assumes an admin user principal of `openidm/admin`.

For more information on setting this property, see `customConfiguration` in the *Connectors Guide*.

`customSensitiveConfiguration`

The password for the user principal.

For more information on setting this property, see `customSensitiveConfiguration` in the *Connectors Guide*.

Your connector configuration should look something like the following:

```
...
"configurationProperties" : {
  "host" : "192.0.2.0",
  "port" : 22,
  "user" : "admin",
  "password" : "Passw0rd",
  "prompt" : "admin@myhost:~$",
  "sudoCommand" : "/usr/bin/sudo",
  "echoOff" : true,
  "terminalType" : "vt102",
  "setLocale" : false,
  "locale" : "en_US.utf8",
  "connectionTimeout" : 5000,
  "expectTimeout" : 5000,
  "authenticationType" : "PASSWORD",
  "throwOperationTimeoutException" : true,
  "customConfiguration" : "kadmin { cmd = '/usr/sbin/kadmin.local'; user='openidm/admin';
default_realm='EXAMPLE.COM' }",
  "customSensitiveConfiguration" : "kadmin { password = 'Passw0rd' }",
...

```

IDM encrypts passwords in the configuration when it starts up, or whenever it reloads the configuration file.

For information about the complete Kerberos connector configuration, see "Configuring the Kerberos Connector" in the *Connectors Guide*.

Caution

Do not modify the value of the `scriptRoots` or `classpath` properties unless you have extracted the scripts from the connector bundle and placed them on the filesystem.

Run the Sample

The commands in this section achieve the following:

1. Start IDM and check that the connector can reach the Kerberos server.
2. Create two users in the managed repository.

3. Reconcile the managed repository with the Kerberos server so that the new users are created in Kerberos.
4. Retrieve the details of one of the new Kerberos principals from the server.
5. Delete one of the managed users.
6. Reconcile the managed repository again and note that the corresponding Kerberos principal has been deleted.
1. Start IDM with the configuration for the Kerberos sample:

```
/path/to/openidm/startup.sh -p samples/sync-with-kerberos
```

2. Test that your connector configuration is correct and that IDM can reach your Kerberos server, with the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=test"
[
  {
    "name": "kerberos",
    "enabled": true,
    "config": "config/provisioner.openicf/kerberos",
    "objectTypes": [
      "__ALL__",
      "account"
    ],
    "connectorRef": {
      "bundleName": "org.forgerock.openicf.connectors.kerberos-connector",
      "connectorName": "org.forgerock.openicf.connectors.kerberos.KerberosConnector",
      "bundleVersion": "[1.4.0.0,1.6.0.0)"
    },
    "displayName": "Kerberos Connector",
    "ok": true
  }
]
```

If the command returns `"ok": true`, as in the preceding output, your configuration is correct and you can continue with the sample.

3. Retrieve a list of the existing user principals in the Kerberos database:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/kerberos/account?_queryId=query-all-ids"
{
  "result": [
    {
```

```

    "_id": "K/M@EXAMPLE.COM",
    "principal": "K/M@EXAMPLE.COM"
  },
  {
    "_id": "kadmin/admin@EXAMPLE.COM",
    "principal": "kadmin/admin@EXAMPLE.COM"
  },
  {
    "_id": "kadmin/changepw@EXAMPLE.COM",
    "principal": "kadmin/changepw@EXAMPLE.COM"
  },
  {
    "_id": "kadmin/krbl.example.com@EXAMPLE.COM",
    "principal": "kadmin/krbl.example.com@EXAMPLE.COM"
  },
  {
    "_id": "kiprop/krbl.example.com@EXAMPLE.COM",
    "principal": "kiprop/krbl.example.com@EXAMPLE.COM"
  },
  {
    "_id": "krbtgt/EXAMPLE.COM@EXAMPLE.COM",
    "principal": "krbtgt/EXAMPLE.COM@EXAMPLE.COM"
  },
  {
    "_id": "openidm/admin@EXAMPLE.COM",
    "principal": "openidm/admin@EXAMPLE.COM"
  }
],
...
}

```

4. Create two new managed users, either over REST or by using the Admin UI.

The following command creates users bjensen and scarter over REST. To create similar users by using the Admin UI, select Managed > User and click New User:

```

curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-type: application/json" \
--request POST \
--data '{
  "userName": "bjensen",
  "givenName": "Barbara",
  "sn": "Jensen",
  "password": "Passw0rd",
  "displayName": "Barbara Jensen",
  "mail": "bjensen@example.com"
}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "_id": "ce3d9b8f-1d15-4950-82c1-f87596aadcb6",
  "_rev": "00000000792afa08",
  "userName": "bjensen",
  "givenName": "Barbara",
  "sn": "Jensen",
  "displayName": "Barbara Jensen",

```

```

"mail": "bjensen@example.com",
"accountStatus": "active",
"effectiveRoles": [],
"effectiveAssignments": []
}

curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-type: application/json" \
--request POST \
--data '{
  "userName": "scarter",
  "givenName": "Steven",
  "sn": "Carter",
  "password": "Passw0rd",
  "displayName": "Steven Carter",
  "mail": "scarter@example.com"
}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "id": "a204ca60-b0fc-42f8-bf93-65bb30131361",
  "_rev": "000000004121fb7e",
  "userName": "scarter",
  "givenName": "Steven",
  "sn": "Carter",
  "displayName": "Steven Carter",
  "mail": "scarter@example.com",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}

```

5. Run a reconciliation operation between the managed user repository and the Kerberos database to create the new users bjensen and scarter in Kerberos. You can run the reconciliation over REST, or using the Admin UI.

The following command creates runs the reconciliation over REST:

```

curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?_action=recon&mapping=managedUser_systemKerberos"
{
  "_id": "862ab9ba-d1d9-4058-b6bc-a23a94b68776-234",
  "state": "ACTIVE"
}

```

To run the reconciliation by using the Admin UI, select Configure > Mappings, click on the [managedUser_systemKerberos](#) mapping, and click Reconcile.

6. Retrieve the list of Kerberos user principals again. You should now see bjensen and scarter in this list:


```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/kerberos/account?_queryId=query-all-ids"
{
  "result": [
    {
      "_id": "bjensen@EXAMPLE.COM",
      "principal": "bjensen@EXAMPLE.COM"
    },
    {
      "_id": "scarter@EXAMPLE.COM",
      "principal": "scarter@EXAMPLE.COM"
    },
    ...
    {
      "_id": "openidm/admin@EXAMPLE.COM",
      "principal": "openidm/admin@EXAMPLE.COM"
    }
  ],
  ...
}
```

7. Retrieve bjensen's complete user principal from the Kerberos server:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/kerberos/account/bjensen@EXAMPLE.COM"
{
  "_id": "bjensen@EXAMPLE.COM",
  "lastFailedAuthentication": "[never]",
  "passwordExpiration": "[none]",
  "lastSuccessfulAuthentication": "[never]",
  "maximumTicketLife": "0 days 10:00:00",
  "lastModified": "Tue May 24 04:05:45 EDT 2016 (openidm/admin@EXAMPLE.COM)",
  "policy": "user [does not exist]",
  "expirationDate": "[never]",
  "failedPasswordAttempts": "0",
  "maximumRenewableLife": "7 days 00:00:00",
  "principal": "bjensen@EXAMPLE.COM",
  "lastPasswordChange": "Tue May 24 04:05:45 EDT 2016"
}
```

Note the default values for properties such as `maximumRenewableLife`. These values are set in your connector configuration. For more information, see "Configuring the Kerberos Connector" in the *Connectors Guide*.

To perform this step in the Admin UI, select Manage > User, click bjensen's entry, and click the Linked Systems tab to display her corresponding entry on the Kerberos server.

8. Delete the managed user bjensen by specifying her managed object ID in the DELETE request.

First, obtain her ID by querying for her userName:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=userName+eq+'bjensen'"
{
  "result": [
    {
      "_id": "ce3d9b8f-1d15-4950-82c1-f87596aadcb6",
      "_rev": "00000000a92657c7",
      "userName": "bjensen",
      "givenName": "Barbara",
      "sn": "Jensen",
      "displayName": "Barbara Jensen",
      "mail": "bjensen@example.com",
      "accountStatus": "active",
      "effectiveRoles": [],
      "effectiveAssignments": []
    }
  ],
  ...
}
```

Now delete the user with ID `ce3d9b8f-1d15-4950-82c1-f87596aadcb6`. This ID will obviously be different in your example.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/managed/user/ce3d9b8f-1d15-4950-82c1-f87596aadcb6"
{
  "_id": "ce3d9b8f-1d15-4950-82c1-f87596aadcb6",
  "_rev": "00000000a92657c7",
  "userName": "bjensen",
  "givenName": "Barbara",
  "sn": "Jensen",
  "displayName": "Barbara Jensen",
  "mail": "bjensen@example.com",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

To delete bjensen's managed user entry by using the Admin UI, select Manage > User, click on bjensen's entry, select the checkbox next to her entry, and click Delete Selected.

9. Reconcile the managed user repository and the Kerberos database again:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?_action=recon&mapping=managedUser_systemKerberos"
{
  "_id": "862ab9ba-d1d9-4058-b6bc-a23a94b68776-584",
  "state": "ACTIVE"
}
```

- Retrieve the list of Kerberos user principals again. The Kerberos principal for bjensen should have been removed from the list:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/kerberos/account?_queryId=query-all-ids"
{
  "result": [
    {
      "_id": "K/M@EXAMPLE.COM",
      "principal": "K/M@EXAMPLE.COM"
    },
    {
      "_id": "kadmin/admin@EXAMPLE.COM",
      "principal": "kadmin/admin@EXAMPLE.COM"
    },
    {
      "_id": "kadmin/angepw@EXAMPLE.COM",
      "principal": "kadmin/angepw@EXAMPLE.COM"
    },
    {
      "_id": "kadmin/krb1.example.com@EXAMPLE.COM",
      "principal": "kadmin/krb1.example.com@EXAMPLE.COM"
    },
    {
      "_id": "kiprop/krb1.example.com@EXAMPLE.COM",
      "principal": "kiprop/krb1.example.com@EXAMPLE.COM"
    },
    {
      "_id": "krbtgt/EXAMPLE.COM@EXAMPLE.COM",
      "principal": "krbtgt/EXAMPLE.COM@EXAMPLE.COM"
    },
    {
      "_id": "scarter@EXAMPLE.COM",
      "principal": "scarter@EXAMPLE.COM"
    },
    {
      "_id": "openidm/admin@EXAMPLE.COM",
      "principal": "openidm/admin@EXAMPLE.COM"
    }
  ],
  ...
}
```

Note

Some user IDs in Kerberos include characters such as a forward slash (/) and an "at sign" (@) that prevent them from being used directly in a REST URL. For example, `openidm/system/kerberos/account/kadmin/admin@EXAMPLE.COM`, where the ID is `kadmin/admin@EXAMPLE.COM`. To retrieve such entries directly over REST, you must URL-encode the Kerberos ID as follows:

```
"http://localhost:8080/openidm/system/kerberos/account/kadmin%2Fadmin%40EXAMPLE.COM"
```

Chapter 14

Store Multiple Passwords For Managed Users

This sample demonstrates how to set up multiple passwords for managed users and how to synchronize separate passwords to different external resources.

Note

You cannot run this sample through the Admin UI. To make the sample work with the Admin UI, set the **viewable** and **required** fields of the **password** property in the **conf/managed.json** file as follows:

```
"password" : {  
  "title" : "Password",  
  "type" : "string",  
  "viewable" : true,  
  ...  
}
```

- "Configure the Multiple Passwords Sample"
- "Password History Policy"
- "LDAP Server Configuration"
- "Show Multiple Accounts"
- "Show the Password History Policy"

Configure the Multiple Passwords Sample

This sample assumes the following scenario:

- The managed/user repository is the source system.
- There are two target LDAP servers — **ldap** and **ldap2**.

For the purposes of this sample, the two servers are represented by two separate organizational units on a single ForgeRock Directory Services (DS) instance.

- Managed user objects have two additional password fields, each mapped to one of the two LDAP servers.
- Both LDAP servers have a requirement for a password history policy, but the history size differs for the two policies.

The sample shows how to extend the password history policy, described in "Creating a Password History Policy" in the *Security Guide*, to apply to multiple password fields.

- The value of a managed user's `password` field is used by default for the additional passwords *unless* the CREATE, UPDATE, or PATCH requests on the managed user explicitly contain a value for these additional passwords.

The sample includes several customized configuration files in the `samples/multiple-passwords/conf/` directory. These customizations are crucial to the sample functionality, and are described in detail in the following list:

`provisioner.openicf-ldap.json`

Configures the connection to the first LDAP directory.

`provisioner.openicf-ldap2.json`

Configures the connection to the second LDAP directory.

`sync.json`

Provides the mappings from the IDM managed user repository to the respective LDAP servers. The file includes two mappings:

- A mapping from IDM managed users to the LDAP user objects at the `system/ldap/account` endpoint. This endpoint represents the `ou=People` subtree.
- A mapping from IDM managed users to the LDAP user objects at the `system/ldap2/account` endpoint. This endpoint represents the `ou=Customers` subtree.

Both mappings include an explicit mapping from `ldapPassword` and `ldap2Password` to `userPassword` in the standard property mappings. Because these passwords are encrypted, a transform script is defined which uses `openidm.decrypt()` to set the value on the target object.

`managed.json`

Contains a customized schema for managed users that includes the additional password fields.

This file has been customized as follows:

- The schema includes an `ldapPassword` field that is mapped to the accounts in the `system/ldap/accounts` target. This field is subject to the standard policies associated with the `password` field of a managed user. In addition, the `ldapPassword` must contain two capital letters instead of the usual one capital letter requirement.
- The schema includes an `ldap2Password` field that is mapped to the accounts in the `system/ldap2/accounts` target. This field is subject to the standard policies associated with the `password` field of a managed user. In addition, the `ldap2Password` must contain two numbers instead of the usual one number requirement.
- A custom password history policy ("`policyId`" : "`is-new`") applies to each of the two mapped password fields `ldapPassword`, and `ldap2Password`.

router.json

A scripted filter on `managed/user` and `policy/managed/user` that populates the values of the additional password fields with the value of the main `password` field if the additional fields are not included in the request content.

The sample includes the following customized scripts in the `script` directory:

- `onCreate-user-custom.js` and `onUpdate-user-custom.js` are used for validation of the password history policy when a user is created or updated.
- `pwpolicy.js` is an additional policy script for the password history policy.
- `set-additional-passwords.js` populates the values of the additional password fields with the value of the main `password` field if the additional fields are not included in the request content.

Password History Policy

The sample includes a custom password history policy. Although the sample demonstrates the history of password attributes only, you can use this policy to enforce history validation on any managed object property.

The following configuration changes set up the password history policy:

- A `fieldHistory` property is added to managed users. The value of this field is a map of field names to a list of historical values for that field. These lists of values are used by the policy to determine if a new value has previously been used.

The `fieldHistory` property is not accessible over REST by default, and cannot be modified.

- The `onCreate-user-custom.js` script performs the standard `onCreate` tasks for a managed user object but also stores the initial value of each of the fields for which IDM should keep a history. The script is passed the following configurable properties:
 - `historyFields` — a list of the fields to store history on.
 - `historySize` — the number of historical fields to store.
- The `onUpdate-user-custom.js` script compares the old and new values of the historical fields on update events to determine if the values have changed. When a new value is detected, it is stored in the list of historical values for that field.

This script also contains logic to deal with the comparison of encrypted field values. The script is passed the following configurable properties:

- `historyFields` — a list of the fields to store history on.
- `historySize` — the number of historical fields to store.

- The `pwpolicy.js` script contains the additional policy definition for the password history policy. This script compares the new field value with the list of historical values for each field.

The policy configuration (`policy.json`) references this script in its `additionalFiles` list, so that the policy service loads the policy definition. The new policy takes a `historyLength` parameter, which indicates the number of historical values to enforce the policy on. This number must not exceed the `historySize` specified in the `onCreate` and `onUpdate` scripts.

- The `ldapPassword` and `ldap2Password` fields in the managed user schema have been updated with the policy. For the purposes of this sample the `historySize` has been set to 2 for `ldapPassword` and to 4 for `ldap2Password`.

LDAP Server Configuration

1. Set up DS using `/path/to/openidm/samples/multiple-passwords/data/Example.ldif`.
2. Perform an `ldapsearch` on the LDAP directory, and take note of the organizational units:

```
/path/to/openssl/bin/ldapsearch \
--port 1636 \
--useSSL \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--hostname localhost \
--baseDN "dc=example,dc=com" \
--bindDN uid=admin \
--bindPassword password \
"ou=" \
ou
dn: ou=People,dc=example,dc=com
ou: People

dn: ou=Customers,dc=example,dc=com
ou: people
ou: Customers
```

The organizational units, `ou=People` and `ou=Customers`, represent the two different target LDAP systems that our mappings point to.

Show Multiple Accounts

This section starts IDM with the sample configuration, then creates a user with multiple passwords, adhering to the different policies in the configured password policy. The section tests that the user was synchronized to two separate LDAP directories, with the different required passwords, and that the user can bind to each of these LDAP directories.

1. Prepare IDM as described in "Prepare IDM", then start the server with the configuration for the multiple passwords sample:


```
cd /path/to/openidm/
./startup.sh -p samples/multiple-passwords
```

2. Create a user, `jdoe`, providing individual values for each of the different password fields, that comply with the three different password policies:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
  "mail": "john.doe@example.com",
  "password": "Secretpw1",
  "ldapPassword": "S3cretPw",
  "ldap2Password": "Secr3tpw1"
}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "_id": "5ce188f6-252b-429e-aad1-4d8754d77de5",
  "_rev": "00000000d2d76089",
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
  "mail": "john.doe@example.com",
  "ldapPassword": {
    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "cipher": "AES/CBC/PKCS5Padding",
        "stableId": "openidm-sym-default",
        "salt": "lkackh...",
        "data": "T0mljk...",
        "keySize": 16,
        "purpose": "idm.password.encryption",
        "iv": "ehSMbdNn...",
        "mac": "PssP0sW..."
      }
    }
  },
  "ldap2Password": {
    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "cipher": "AES/CBC/PKCS5Padding",
        "stableId": "openidm-sym-default",
        "salt": "lSzMTU54...",
        "data": "UWLQo5Ws...",
        "keySize": 16,
        "purpose": "idm.password.encryption",
        "iv": "ehSMbdNn...",
        "mac": "PssP0s..."
      }
    }
  }
}
```

```

    }
  },
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": [],
  "roles": []
}

```

The user has been created with three different passwords that comply with three distinct password policies. The passwords have been encrypted as defined in the `managed.json` file.

Note that in this example, the user has been created with ID `5ce188f6-252b-429e-aad1-4d8754d77de5`. You will need the user ID when you update the entry later in this procedure.

3. As a result of implicit synchronization, two separate LDAP accounts should have been created for user `jdoe` on our two simulated LDAP servers. For more information about implicit synchronization, see "Types of Synchronization" in the *Synchronization Guide*.
4. Query the IDs in the LDAP directory as follows:

```

curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account?_queryId=query-all-ids"
{
  "result": [
    {
      "_id": "00452010-a164-4065-9f84-3e4636a3ee20",
    },
    {
      "_id": "e5b35587-2d7c-4faa-b3e5-962f5a4ada5c",
    }
  ],
  ...
}

```

Note that `jdoe` has two entries—one in `ou=People` and one in `ou=Customers`.

5. To verify the passwords were propagated correctly, perform an LDAP search, bound using each of the `jdoe` accounts, against the `rootDSE`.

Note

For the following commands, make sure to enter 2 or 3 at the following prompt:

```
Do you trust this server certificate?
```

- 1) No
- 2) Yes, for this session only
- 3) Yes, also add it to a truststore
- 4) View certificate details

```
Enter choice: [1]: 2
```

```
/path/to/openssl/bin/ldapssearch \
--hostname localhost \
--port 1636 \
--useSSL \
--bindDN uid=jdoe,ou=People,dc=example,dc=com \
--bindPassword S3cretPw \
--searchScope base \
--baseDN "" "(objectClass=*)"
dn:
objectClass: top
objectClass: ds-root-dse
```

```
/path/to/openssl/bin/ldapssearch \
--hostname localhost \
--port 1636 \
--useSSL \
--bindDN uid=jdoe,ou=Customers,dc=example,dc=com \
--bindPassword Secr3tpw1 \
--searchScope base \
--baseDN "" "(objectClass=*)"
dn:
objectClass: top
objectClass: ds-root-dse
```

6. Patch jdoe's managed user entry (5ce188f6-252b-429e-aad1-4d8754d77de5) to change his `ldapPassword`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
  "operation": "replace",
  "field": "ldapPassword",
  "value": "TTestw0rd"
} ]' \
"http://localhost:8080/openidm/managed/user/5ce188f6-252b-429e-aad1-4d8754d77de5"
{
  "_id": "5ce188f6-252b-429e-aad1-4d8754d77de5",
  "_rev": "000000001298f6a6",
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
  ...
  "ldapPassword": {
```

```
"$crypto": {
  "type": "x-simple-encryption",
  "value": {
    "cipher": "AES/CBC/PKCS5Padding",
    "stableId": "openidm-sym-default",
    "salt": "Vlco8e...",
    "data": "INj9lk...",
    "keySize": 16,
    "purpose": "idm.password.encryption",
    "iv": "ehSMbdNn...",
    "mac": "PssP0sW..."
  }
},
...
}
```

- To verify the password change propagated correctly, perform an LDAP search, bound using `jdoe` from the `People` organizational unit, against the `rootDSE`.

Note

For the following command, make sure to enter **2** or **3** at the following prompt:

```
Do you trust this server certificate?

1) No
2) Yes, for this session only
3) Yes, also add it to a truststore
4) View certificate details

Enter choice: [1]: 2
```

```
/path/to/openssl/bin/ldapsrch \
--hostname localhost \
--port 1636 \
--useSSL \
--bindDN uid=jdoe,ou=People,dc=example,dc=com \
--bindPassword TTestw0rd \
--searchScope base \
--baseDN "" "(objectClass=*)"
dn:
objectClass: top
objectClass: ds-root-dse
```

Show the Password History Policy

This section demonstrates the password history policy by patching `jdoe`'s managed user entry, changing his `ldapPassword` multiple times.

- Send the following patch requests, changing the value of `jdoe`'s `ldapPassword` each time:

```
curl \
```

```
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
  "operation": "replace",
  "field": "ldapPassword",
  "value": "TTestw0rd1"
} ]' \
"http://localhost:8080/openidm/managed/user/5ce188f6-252b-429e-aad1-4d8754d77de5"
{
  "_id": "5ce188f6-252b-429e-aad1-4d8754d77de5",
  "_rev": "00000000a92657c7",
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
  "mail": "john.doe@example.com",
  ...
  "ldapPassword": {
    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "cipher": "AES/CBC/PKCS5Padding",
        "stableId": "openidm-sym-default",
        "salt": "TjollL7...",
        "data": "Unbaloo...",
        "keySize": 16,
        "purpose": "idm.password.encryption",
        "iv": "ehSMbdNn...",
        "mac": "PssP0sW..."
      }
    }
  },
  ...
}
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
  "operation": "replace",
  "field": "ldapPassword",
  "value": "TTestw0rd2"
} ]' \
"http://localhost:8080/openidm/managed/user/5ce188f6-252b-429e-aad1-4d8754d77de5"
{
  "_id": "5ce188f6-252b-429e-aad1-4d8754d77de5",
  "_rev": "00000000dc6160c8",
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
  ...
  "ldapPassword": {
```

```

    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "cipher": "AES/CBC/PKCS5Padding",
        "stableId": "openidm-sym-default",
        "salt": "Ynio9n...",
        "data": "R0o12b...",
        "keySize": 16,
        "purpose": "idm.password.encryption",
        "iv": "ehSMbdNn...",
        "mac": "PssP0sW..."
      }
    }
  },
  ...
}

```

```

curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
  "operation": "replace",
  "field": "ldapPassword",
  "value": "TTestw0rd3"
} ]' \
"http://localhost:8080/openidm/managed/user/5ce188f6-252b-429e-aad1-4d8754d77de5"
{
  "_id": "5ce188f6-252b-429e-aad1-4d8754d77de5",
  "_rev": "00000000a92657c7",
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
  ...
  "ldapPassword": {
    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "cipher": "AES/CBC/PKCS5Padding",
        "stableId": "openidm-sym-default",
        "salt": "9kilajT...",
        "data": "Hnkja98...",
        "keySize": 16,
        "purpose": "idm.password.encryption",
        "iv": "ehSMbdNn...",
        "mac": "PssP0sW..."
      }
    }
  }
},
  ...
}

```

User jdoe now has a *history* of `ldapPassword` values, that contains `TTestw0rd3`, `TTestw0rd2`, `TTestw0rd1`, and `TTestw0rd`, in that order.

- The history size for the `ldapPassword` policy is set to 2. To demonstrate the history policy, attempt to patch `jdoe`'s entry with a password value that was used in his previous 2 password changes:

`TTestw0rd2`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
  "operation": "replace",
  "field": "ldapPassword",
  "value": "TTestw0rd2"
} ]' \
"http://localhost:8080/openidm/managed/user/5ce188f6-252b-429e-aad1-4d8754d77de5"
{
  "code": 403,
  "reason": "Forbidden",
  "message": "Failed policy validation",
  "detail": {
    "result": false,
    "failedPolicyRequirements": [
      {
        "policyRequirements": [
          {
            "policyRequirement": "IS_NEW"
          }
        ],
        "property": "ldapPassword"
      }
    ]
  }
}
```

The password change fails the `IS_NEW` policy requirement.

- Change `jdoe`'s `ldapPassword` to a value not used in the previous two updates:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
  "operation": "replace",
  "field": "ldapPassword",
  "value": "TTestw0rd"
} ]' \
"http://localhost:8080/openidm/managed/user/5ce188f6-252b-429e-aad1-4d8754d77de5"
{
  "_id": "5ce188f6-252b-429e-aad1-4d8754d77de5",
  "_rev": "00000000792afa08",
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
}
```

```
...
"ldapPassword": {
  "$crypto": {
    "type": "x-simple-encryption",
    "value": {
      "cipher": "AES/CBC/PKCS5Padding",
      "stableId": "openidm-sym-default",
      "salt": "Ivmal5...",
      "data": "0mkywe...",
      "keySize": 16,
      "purpose": "idm.password.encryption",
      "iv": "ehSMbdNn...",
      "mac": "PssP0sW..."
    }
  }
},
...
```

The password change succeeds.

Chapter 15

Link Multiple Accounts to a Single Identity

This sample illustrates how IDM handles links from multiple accounts to a single identity.

The sample is based on a common use case in the insurance industry, where a company (Example.com) employs agents to sell policies to their insured customers. Most of their agents are also insured, which means that they have two distinct *roles* within the company - customers, and agents. With minor changes, this sample works for other use cases. For example, a hospital employs doctors who treat patients, and some of those same doctors are also patients of the hospital.

Sample Overview

You define mappings between source and target accounts in the your project's `sync.json` file. As part of a mapping, you can create a link between a single source account and multiple target accounts using a *link qualifier*, that enables one-to-many relationships in mappings and policies. For more information about mappings and link qualifiers, see "*Mapping Data Between Resources*" in the *Synchronization Guide* and "Map a Single Source Object to Multiple Target Objects" in the *Synchronization Guide*.

This sample uses two link qualifiers:

- **Insured** represents the accounts associated with Example.com's Insured customers, created under the LDAP container `ou=Customers,dc=example,dc=com`.
- **Agent** represents agent accounts, considered independent contractors, and created under the LDAP container `ou=Contractors,dc=example,dc=com`.

Assume that agents and insured customers connect using two different portals, and that each group has access to different features, depending on the portal.

Agents might have two separate accounts; one each for professional and personal use. Although the accounts are different, the identity information for each agent should be the same for both accounts.

This sample therefore uses link qualifiers to distinguish the two *categories* of users. The link qualifiers are named **insured** and **agent**, and are defined as part of the `managedUser_systemLdapAccounts` mapping in the `sync.json` file:

```
{
  "name" : "managedUser_systemLdapAccounts",
  "source" : "managed/user",
  "target" : "system/ldap/account",
  "linkQualifiers" : [
    "insured",
    "agent"
  ],
  ...
}
```

You can check this configuration in the Admin UI. Click Configure > Mappings > `managedUser_systemLdapAccounts` > Properties > Link Qualifiers. You should see `insured` and `agent` in the list of configured link qualifiers.

In addition, the sample uses a transformation script that determines the LDAP Distinguished Name (`dn`) from the user category. The following excerpt of the `sync.json` file shows that script:

```
{
  "target" : "dn",
  "transform" : {
    "type" : "text/javascript",
    "globals" : { },
    "source" :
      "if (linkQualifier === 'agent') {
        'uid=' + source.userName + ',ou=Contractors,dc=example,dc=com';
      } else if (linkQualifier === 'insured') {
        'uid=' + source.userName + ',ou=Customers,dc=example,dc=com';
      }"
  },
}
```

Finally, the following `validSource` script assesses the effective roles of a managed user to determine if that user has an `Agent` or `Insured` role. The script then assigns a link qualifier based on the assessed role.

```
"validSource" : {
  "type" : "text/javascript",
  "globals" : { },
  "source" : "var res = false;
  var i=0;

  while (!res && i < source.effectiveRoles.length) {
    var roleId = source.effectiveRoles[i];
    if (roleId != null && roleId.indexOf("/") != -1) {
      var roleInfo = openidm.read(roleId);
      logger.warn("Role Info : {}",roleInfo);
      res = (((roleInfo.properties.name === 'Agent')
        &&(linkQualifier ==='agent'))
        || ((roleInfo.properties.name === 'Insured')
        &&(linkQualifier ==='insured')));
    }
    i++;
  }
  res"
}
```

Prepare the Sample

1. Set up DS using `/path/to/openidm/samples/multi-account-linking/data/Example.ldif`.
2. Prepare IDM, and start the server using the sample configuration:

```
cd /path/to/openidm/  
./startup.sh -p samples/multi-account-linking
```

Run the Sample

Create the Users, Roles, and Assignments

1. Create the managed users for John Doe and Barbara Jensen.

To set up these managed users using the Admin UI, select Manage > User > New User; otherwise, using the REST interface:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "displayName": "Barbara Jensen",
  "description": "Created for OpenIDM",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "telephoneNumber": "1-360-229-7105",
  "sn": "Jensen",
  "userName": "bjensen",
  "accountStatus": "active"
}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "_id": "580f1441-ff8e-434b-9605-90e10a6fbdf6",
  "_rev": "00000000792afa08",
  "displayName": "Barbara Jensen",
  "description": "Created for OpenIDM",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "telephoneNumber": "1-360-229-7105",
  "sn": "Jensen",
  "userName": "bjensen",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "displayName": "John Doe",
  "description": "Created for OpenIDM",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "1-415-599-1100",
  "sn": "Doe",
  "userName": "jdoe",
  "accountStatus": "active"
}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "_id": "02632173-e413-4af1-8495-f749d5880226",
  "_rev": "000000001298f6a6",
  "displayName": "John Doe",
  "description": "Created for OpenIDM",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "1-415-599-1100",
  "sn": "Doe",
  "userName": "jdoe",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

Note

Make sure to record the unique `_id` for both managed users.

2. Set up the managed roles Agent and Insured, to distinguish between the two user types.

To set up these roles using the Admin UI, select Manage > Role > New Role; otherwise, using the REST interface:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "name": "Agent",
  "description": "Role assigned to insurance agents."
}' \
"http://localhost:8080/openidm/managed/role?_action=create"
{
  "_id": "1b58ec8d-fae2-4b28-a5cf-b63567e4cf3f",
  "_rev": "000000005b3d5ebd",
  "name": "Agent",
  "description": "Role assigned to insurance agents."
}
```

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "name": "Insured",
  "description": "Role assigned to insured customers."
}' \
"http://localhost:8080/openidm/managed/role?_action=create"
{
  "_id": "617368f2-fa4e-44a2-a25a-f0a86e16ef00",
  "_rev": "000000002b845f24",
  "name": "Insured",
  "description": "Role assigned to insured customers."
}
```

Note

Make sure to record the unique `_id` for both managed roles.

3. Grant the managed roles to the users. In this sample, `jdoe` is an agent and customer, and `bjensen` is only a customer.

To grant the roles, you need the `_ids` you recorded when you created the users and roles.

- a. The following command grants the Agent role to `jdoe`:

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
{
  "name": "Agent",
  "description": "Role assigned to insurance agents."
}]'
```

```

    "operation": "add",
    "field": "/roles/-",
    "value": {
      "_ref": "managed/role/1b58ec8d-fae2-4b28-a5cf-b63567e4cf3f"
    }
  }
] \
"http://localhost:8080/openidm/managed/user/02632173-e413-4af1-8495-f749d5880226"
{
  "_id": "02632173-e413-4af1-8495-f749d5880226",
  "_rev": "00000000dc6160c8",
  "displayName": "John Doe",
  "description": "Created for OpenIDM",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "1-415-599-1100",
  "sn": "Doe",
  "userName": "jdoe",
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": [
    {
      "_refResourceCollection": "managed/role",
      "_refResourceId": "1b58ec8d-fae2-4b28-a5cf-b63567e4cf3f",
      "_ref": "managed/role/1b58ec8d-fae2-4b28-a5cf-b63567e4cf3f"
    }
  ]
}

```

- b. The following command grants the Insured role to user bjensen:

```

curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
{
  "operation": "add",
  "field": "/roles/-",
  "value": {
    "_ref": "managed/role/617368f2-fa4e-44a2-a25a-f0a86e16ef00"
  }
}
] \
"http://localhost:8080/openidm/managed/user/580f1441-ff8e-434b-9605-90e10a6fbd6"
{
  "_id": "580f1441-ff8e-434b-9605-90e10a6fbd6",
  "_rev": "000000004cab60c8",
  "displayName": "Barbara Jensen",
  "description": "Created for OpenIDM",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "telephoneNumber": "1-360-229-7105",
  "sn": "Jensen",
  "userName": "bjensen",
  "accountStatus": "active",

```

```
"effectiveAssignments": [],
"effectiveRoles": [
  {
    "_refResourceCollection": "managed/role",
    "_refResourceId": "617368f2-fa4e-44a2-a25a-f0a86e16ef00",
    "_ref": "managed/role/617368f2-fa4e-44a2-a25a-f0a86e16ef00"
  }
]
```

- c. The following command grants the Insured role to jdoe:

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/roles/-",
    "value": {
      "_ref": "managed/role/617368f2-fa4e-44a2-a25a-f0a86e16ef00"
    }
  }
]' \
"http://localhost:8080/openidm/managed/user/02632173-e413-4af1-8495-f749d5880226"
{
  "_id": "02632173-e413-4af1-8495-f749d5880226",
  "_rev": "00000000a92657c7",
  "_displayName": "John Doe",
  "description": "Created for OpenIDM",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "1-415-599-1100",
  "sn": "Doe",
  "userName": "jdoe",
  "accountStatus": "active",
  "effectiveAssignments": []
  "effectiveRoles": [
    {
      "_refResourceCollection": "managed/role",
      "_refResourceId": "1b58ec8d-fae2-4b28-a5cf-b63567e4cf3f",
      "_ref": "managed/role/1b58ec8d-fae2-4b28-a5cf-b63567e4cf3f"
    },
    {
      "_refResourceCollection": "managed/role",
      "_refResourceId": "617368f2-fa4e-44a2-a25a-f0a86e16ef00",
      "_ref": "managed/role/617368f2-fa4e-44a2-a25a-f0a86e16ef00"
    }
  ]
}
```

Notice jdoe now has two managed roles, as shown by the multiple `effectiveRoles`.

4. Create the managed assignments.

Assignments specify what a role actually does on a target system. A single account frequently has different functions on a system. For example, while agents might be members of the Contractor group, insured customers might be part of a Chat Users group (possibly for access to customer service). The following commands create two managed assignments that will be attached to the agent and insured roles. Note the `_id` of each assignment because you will need these when you attach the assignment to its corresponding role.

The following command creates an `ldapAgent` assignment. Users who have this assignment will have their `ldapGroups` property in DS set to `cn=Contractors,ou=Groups,dc=example,dc=com`. The assignment is associated with the `agent` link qualifier:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "name": "ldapAgent",
  "description": "LDAP Agent Assignment",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "ldapGroups",
      "value": [
        "cn=Contractors,ou=Groups,dc=example,dc=com"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ],
  "linkQualifiers": ["agent"]
}' \
"http://localhost:8080/openidm/managed/assignment?_action=create"
{
  "_id": "cc0dbcdc-64a4-4f5b-aade-648fc012e2b5",
  "_rev": "00000000c7554e13",
  "name": "ldapAgent",
  "description": "LDAP Agent Assignment",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "ldapGroups",
      "value": [
        "cn=Contractors,ou=Groups,dc=example,dc=com"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ],
  "linkQualifiers": [
    "agent"
  ]
}
```


The following command creates an `ldapCustomer` assignment. Users who have this assignment will have their `ldapGroups` property in DS set to `cn=Chat Users,ou=Groups,dc=example,dc=com`. The assignment is associated with the `insured` link qualifier:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "name": "ldapCustomer",
  "description": "LDAP Customer Assignment",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "ldapGroups",
      "value": [
        "cn=Chat Users,ou=Groups,dc=example,dc=com"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ],
  "linkQualifiers": ["insured"]
}' \
"http://localhost:8080/openidm/managed/assignment?_action=create"
{
  "_id": "56b1f300-7156-4110-9b23-2052c16dd2aa",
  "_rev": "000000000cde398e",
  "name": "ldapCustomer",
  "description": "LDAP Customer Assignment",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "ldapGroups",
      "value": [
        "cn=Chat Users,ou=Groups,dc=example,dc=com"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ],
  "linkQualifiers": [
    "insured"
  ]
}
```

5. Add the assignments to their respective roles.

Add the `ldapCustomer` assignment to the Insured customer role:

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
{
  "operation": "add",
  "field": "/assignments/-",
  "value": {
    "_ref": "managed/assignment/56b1f300-7156-4110-9b23-2052c16dd2aa"
  }
}
]' \
"http://localhost:8080/openidm/managed/role/617368f2-fa4e-44a2-a25a-f0a86e16ef00"
{
  "_id": "617368f2-fa4e-44a2-a25a-f0a86e16ef00",
  "_rev": "0000000050c62938",
  "name": "Insured",
  "description": "Role assigned to insured customers."
}
```

6. Add the `ldapAgent` assignment to the Agent role:

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
{
  "operation": "add",
  "field": "/assignments/-",
  "value": {
    "_ref": "managed/assignment/cc0dbcdc-64a4-4f5b-aade-648fc012e2b5"
  }
}
]' \
"http://localhost:8080/openidm/managed/role/1b58ec8d-fae2-4b28-a5cf-b63567e4cf3f"
{
  "_id": "1b58ec8d-fae2-4b28-a5cf-b63567e4cf3f",
  "_rev": "0000000013e50a6b",
  "name": "Agent",
  "description": "Role assigned to insurance agents."
}
```

Reconcile Managed Users to the LDAP Server

1. With the managed roles and assignments set up, reconcile the managed user repository with the DS data store:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?_action=recon&mapping=managedUser_systemLdapAccounts"
{
  "_id": "a6b46fc6-0731-47d8-83b5-89cca8963512-11550",
  "state": "ACTIVE"
}
```

This reconciliation creates three new accounts in DS:

- Two accounts under `ou=Customers,dc=example,dc=com` (one for each user who has the insured customers role), `bjensen` and `jdoe`.
- One account under `ou=Contractors,dc=example,dc=com` (for the use who has the agents role), `jdoe`.

Note that both of these users already existed in DS (from the `Example.ldif` file that you imported during the setup).

2. Query the list of users in DS to see the multiple accounts created for `jdoe` and `bjensen` as a result of the reconciliation:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account?_queryId=query-all-ids"
{
  "result": [
    {
      "_id": "3bbf1f43-e120-4d34-a4c9-05bd02be23bd"
    },
    {
      "_id": "d6c73ea1-fd05-4b80-8625-c50303755c91"
    },
    {
      "_id": "0acc77a5-0f38-473b-b533-e37ca1d4fd4c"
    },
    {
      "_id": "3310b29a-0d7f-4ed5-aa0d-795d2780e002"
    },
    {
      "_id": "3c8e3c3d-f748-44c1-8cfc-172f5b0a9b5e"
    }
  ],
  ...
}
```

Chapter 16

Link Historical Accounts

This sample demonstrates the retention of inactive (historical) LDAP accounts that have been linked to a corresponding managed user account. The sample builds on "*Two Way Synchronization Between LDAP and IDM*" and uses the LDAP connector to connect to a ForgeRock Directory Services (DS) instance. You can use any LDAP-v3 compliant directory server.

Sample Overview

In this sample, IDM is the source resource. Managed users in the IDM repository maintain a list of the accounts to which they have been linked on the local LDAP server. This list is stored in the `historicalAccounts` field of the managed user entry. The list contains a reference to all past and current LDAP accounts. Each LDAP account in the list is represented as a `relationship` and includes information about the date the accounts were linked or unlinked, and whether the account is currently active.

This sample includes the following custom scripts, in its `script` directory:

- `onLink-managedUser_systemLdapAccounts.js`

When a managed user object is linked to a target LDAP object, this script creates the relationship entry in the managed user's `historicalAccounts` property. The script adds two relationship properties:

- `linkDate` — specifies the date that the link was created.
 - `active` — boolean true/false. When set to true, this property indicates that the target object is *currently* linked to the managed user account.
- `onUnlink-managedUser_systemLdapAccounts.js`

When a managed user object is unlinked from a target LDAP object, this script updates that relationship entry's properties with an `unlinkDate` that specifies when the target was unlinked, and sets the `active` property to false, indicating that the target object is no longer linked.

- `check_account_state_change.js`

During liveSync or reconciliation, this script checks if the LDAP account state has changed. If the state has changed, the script updates the historical account properties to indicate the new state (enabled or disabled), and the date that the state was changed. This date can only be approximated and is set to the time that the change was detected by the script.

- `LdapBackCorrelationQuery.js`

This script correlates entries in the LDAP directory with managed user identities in IDM.

Run the Sample

This section walks you through each step of the sample to demonstrate how historical accounts are stored.

1. Set up DS using `/path/to/openidm/samples/historical-account-linking/data/Example.ldif`.
2. Prepare IDM, and start the server using the sample configuration:

```
cd /path/to/openidm/
./startup.sh -p samples/historical-account-linking
```

3. Create a user, Joe Smith, in IDM:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "userName": "joe.smith",
  "givenName": "Joe",
  "sn": "Smith",
  "password": "Passw0rd",
  "displayName": "Joe Smith",
  "mail": "joe.smith@example.com"
}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "_id": "24356bf0-f026-4dc1-9f68-2a571b0a236f",
  "_rev": "00000000c8dc2137",
  "userName": "joe.smith",
  "givenName": "Joe",
  "sn": "Smith",
  "displayName": "Joe Smith",
  "mail": "joe.smith@example.com",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

Record Joe Smith's system-generated `_id`.

4. Verify that the user Joe Smith was created in DS.

Because implicit synchronization is enabled by default, any change to the managed/user repository should be propagated to DS. For more information about implicit synchronization, see "Types of Synchronization" in the *Synchronization Guide*.

The following command returns all users in DS and shows that user joesmith was created successfully:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account?_queryFilter=true&_fields=_id,dn"
{
  "result": [
    {
      "_id": "0da50512-79bb-3461-bd04-241ee4c785bf",
      "dn": "uid=jdoe,ou=People,dc=example,dc=com"
    },
    {
      "_id": "887732e8-3db2-31bb-b329-20cd6fcecc05",
      "dn": "uid=bjensen,ou=People,dc=example,dc=com"
    },
    {
      "_id": "da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa",
      "dn": "uid=joe.smith0,ou=People,dc=example,dc=com"
    }
  ],
  ...
}
```

Note that Joe Smith's `uid` in DS is appended with a `0`. The `onCreate` script, defined in the mapping (`sync.json`), increments the `uid` each time a new DS entry is linked to the same managed user object.

5. Verify that the historical account *relationship object* that corresponds to this linked LDAP account was created in the IDM repository.

The following command queries Joe Smith's managed user entry and returns all of the `historicalAccounts` for that entry:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/24356bf0-f026-4dc1-9f68-2a571b0a236f/historicalAccounts?_queryFilter=true"
{
  "result": [
    {
      "_id": "3f193422-156b-4b66-adcf-447db1b7d770",
      "_rev": "00000000c2beced4",
      "_ref": "system/ldap/account/da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa",
      "_refResourceCollection": "system/ldap/account",
      "_refResourceId": "da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa",
      "_refProperties": {
        "active": true,
        "stateLastChanged": "Mon May 18 2020 13:47:18 GMT+0200 (SAST)",
        "state": "enabled",
        "linkDate": "Mon May 18 2020 13:47:18 GMT+0200 (SAST)",
        "_id": "3f193422-156b-4b66-adcf-447db1b7d770",
        "_rev": "00000000c2beced4"
      }
    }
  ],
  ...
}
```

At this stage, Joe Smith has only one historical account link — the link to `system/ldap/account/da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa`, which corresponds to the DN `"dn": "uid=joe.smith0,ou=People,dc=example,dc=com"`. Note that the relationship properties (`_refProperties`) show the following information about the linked accounts:

- The date on which the accounts were linked
 - The fact that this link is currently active
 - The state of the account in DS (`enabled`)
6. Enable the liveSync schedule to propagate changes made in DS to the managed user repository.

To start liveSync, set `enabled` to `true` in the `conf/schedule-liveSync.json` file:

```
more /path/to/openidm/samples/historical-account-linking/conf/schedule-liveSync.json
{
  "enabled" : true,
  "type" : "simple",
  "repeatInterval" : 15000,
  ...
}
```

7. Use the `manage-account` command in the `opendj/bin` directory to disable Joe Smith's account in DS:

```
/path/to/openssl/bin/manage-account set-account-is-disabled \
--port 4444 \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--hostname localhost \
--bindDN uid=admin \
--bindPassword password \
--operationValue true \
--targetDN uid=joe.smith0,ou=people,dc=example,dc=com
Account Is Disabled: true
```

Within 15 seconds, according to the configured schedule, liveSync should pick up the change. IDM should then adjust the **state** property in Joe Smith's managed user account.

8. To make sure that the linked account state has changed, request Joe Smith's historical accounts:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/24356bf0-f026-4dc1-9f68-2a571b0a236f/historicalAccounts?queryFilter=true"
{
  "result": [
    {
      "_id": "3f193422-156b-4b66-adcf-447db1b7d770",
      "_rev": "00000000d430e15a",
      "_ref": "system/ldap/account/da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa",
      "_refResourceCollection": "system/ldap/account",
      "_refResourceId": "da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa",
      "_refProperties": {
        "active": true,
        "stateLastChanged": "Mon May 18 2020 13:51:06 GMT+0200 (SAST)",
        "state": "disabled",
        "linkDate": "Mon May 18 2020 13:47:18 GMT+0200 (SAST)",
        "_id": "3f193422-156b-4b66-adcf-447db1b7d770",
        "_rev": "00000000d430e15a"
      }
    }
  ],
  ...
}
```

9. Now, deactivate Joe Smith's managed user account by setting his **accountStatus** property to **inactive**.

You can deactivate the account over the REST interface, or by using the Admin UI.

To use the Admin UI, simply select Manage > User, select Joe Smith's account and change his Status to inactive, on his Details tab.

The following command deactivates Joe Smith's account over REST:


```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  { "operation": "replace",
    "field": "accountStatus",
    "value": "inactive" }
]' \
"http://localhost:8080/openidm/managed/user/24356bf0-f026-4dc1-9f68-2a571b0a236f"
{
  "_id": "24356bf0-f026-4dc1-9f68-2a571b0a236f",
  "_rev": "000000004cc82207",
  "userName": "joe.smith",
  "givenName": "Joe",
  "sn": "Smith",
  "displayName": "Joe Smith",
  "mail": "joe.smith@example.com",
  "accountStatus": "inactive",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

10. Request Joe Smith's historical accounts:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/24356bf0-f026-4dc1-9f68-2a571b0a236f/historicalAccounts?queryFilter=true"
{
  "result": [
    {
      "_id": "3f193422-156b-4b66-adcf-447db1b7d770",
      "_rev": "0000000037beefe7",
      "_ref": "system/ldap/account/da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa",
      "_refResourceCollection": "system/ldap/account",
      "_refResourceId": "da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa",
      "_refProperties": {
        "active": false,
        "stateLastChanged": "Mon May 18 2020 13:51:06 GMT+0200 (SAST)",
        "state": "disabled",
        "linkDate": "Mon May 18 2020 13:47:18 GMT+0200 (SAST)",
        "unlinkDate": "Mon May 18 2020 13:52:33 GMT+0200 (SAST)",
        "_id": "3f193422-156b-4b66-adcf-447db1b7d770",
        "_rev": "0000000037beefe7"
      }
    }
  ]
  ...
}
```

11. Activate Joe Smith's managed user account by setting his `accountStatus` property to active. This action should create a new entry in DS (with `uid=joe.smith1`), and a new link from Joe Smith's managed user object to that DS entry.

You can activate the account over the REST interface, or by using the Admin UI, as described previously.

The following command activates Joe Smith's account over REST:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  { "operation": "replace",
    "field": "accountStatus",
    "value": "active" }
]' \
"http://localhost:8080/openidm/managed/user/24356bf0-f026-4dc1-9f68-2a571b0a236f"
{
  "_id": "24356bf0-f026-4dc1-9f68-2a571b0a236f",
  "_rev": "00000000c8d52133",
  "userName": "joe.smith",
  "givenName": "Joe",
  "sn": "Smith",
  "displayName": "Joe Smith",
  "mail": "joe.smith@example.com",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

12. Verify that a new LDAP entry for user Joe Smith was created in DS.

The following command returns all IDs in DS and shows that two entries now exist for Joe Smith - `uid=joe.smith0` and `uid=joe.smith1`.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account?_queryFilter=true&_fields=_id,dn"
{
  "result": [
    {
      "_id": "0da50512-79bb-3461-bd04-241ee4c785bf",
      "dn": "uid=jdoe,ou=People,dc=example,dc=com"
    },
    {
      "_id": "887732e8-3db2-31bb-b329-20cd6fcecc05",
      "dn": "uid=bjensen,ou=People,dc=example,dc=com"
    },
    {
      "_id": "da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa",
      "dn": "uid=joe.smith0,ou=People,dc=example,dc=com"
    },
    {
      "_id": "52821eec-e00d-4321-8857-f46a870afc45",
      "dn": "uid=joe.smith1,ou=People,dc=example,dc=com"
    }
  ],
  ...
}
```

13. Request Joe Smith's historical accounts:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/24356bf0-f026-4dc1-9f68-2a571b0a236f/historicalAccounts?_queryFilter=true"
{
  "result": [
    {
      "_id": "3f193422-156b-4b66-adcf-447db1b7d770",
      "_rev": "0000000037beefe7",
      "_ref": "system/ldap/account/da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa",
      "_refResourceCollection": "system/ldap/account",
      "_refResourceId": "da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa",
      "_refProperties": {
        "active": false,
        "stateLastChanged": "Mon May 18 2020 13:51:06 GMT+0200 (SAST)",
        "state": "disabled",
        "linkDate": "Mon May 18 2020 13:47:18 GMT+0200 (SAST)",
        "unlinkDate": "Mon May 18 2020 13:52:33 GMT+0200 (SAST)",
        "_id": "3f193422-156b-4b66-adcf-447db1b7d770",
        "_rev": "0000000037beefe7"
      }
    },
    {
      "_id": "8850640c-2233-4ddc-9725-6b4b2d59605f",
      "_rev": "00000000843ce68",
    }
  ],
}
```

```
"_ref": "system/ldap/account/52821eec-e00d-4321-8857-f46a870afc45",
"_refResourceCollection": "system/ldap/account",
"_refResourceId": "52821eec-e00d-4321-8857-f46a870afc45",
"_refProperties": {
  "active": true,
  "stateLastChanged": "Mon May 18 2020 13:54:52 GMT+0200 (SAST)",
  "state": "enabled",
  "linkDate": "Mon May 18 2020 13:54:52 GMT+0200 (SAST)",
  "_id": "8850640c-2233-4ddc-9725-6b4b2d59605f",
  "_rev": "000000000843ce68"
}
},
...
}
```

Note that Joe Smith's entry now shows two DS accounts, but that only the link to `uid=joe.smith1 ("_ref": "system/ldap/account/52821eec-e00d-4321-8857-f46a870afc45",)` is enabled and active.

Chapter 17

Provision Users With Roles

This sample demonstrates how attributes are provisioned to an external system (an LDAP directory), based on role membership. This sample uses ForgeRock Directory Services (DS) as the LDAP directory, but you can use any LDAP v3-compliant server.

Sample Overview

IDM supports two types of roles:

- *Provisioning roles* specify how objects are provisioned to an external system.
- *Authorization roles* specify the authorization rights of a managed object internally, within IDM.

Both provisioning roles and authorization roles use [relationships](#) in the *Object Modeling Guide* to link the role object, and the managed object to which the role applies. For information about managing roles, see "Managed Roles" in the *Object Modeling Guide*.

Important

Most of the commands in this sample can be run by using the command-line but it is generally much easier to use the Admin UI. In some cases, the command-line version makes it easier to explain what is happening in IDM. Therefore, in all steps, the sample first shows the command-line version, and then provides the equivalent method of running the command in the Admin UI.

The main purpose of IDM roles is to provision a set of attributes, based on a managed user's role membership.

The sample assumes a company, example.com. As an *Employee* of example.com, a user should be added to two groups in DS - the Employees group and the Chat Users group (presumably to access certain internal applications). As a *Contractor*, a user should be added only to the Contractors group in DS. A user's employee type must also be set correctly in DS, based on the role that is granted to the user.

Prepare the Sample

Configure the LDAP server as shown in "LDAP Server Configuration". The LDAP user must have write access to create users from IDM on the LDAP server. When you set up the LDAP server, import the LDIF file for this sample ([openidm/samples/provisioning-with-roles/data/Example.ldif](#)).

This section sets up the scenario by performing the following tasks:

1. Start IDM with the configuration for this sample.
2. Create two managed roles, Employee and Contractor.
3. Reconcile the managed user repository with the user entries in the LDAP server.

1. Prepare IDM, and start the server using the sample configuration:

```
cd /path/to/openidm/  
./startup.sh -p samples/provisioning-with-roles
```

2. Create two managed roles, Employee and Contractor, either by using the Admin UI, or by running the following commands:

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "name": "Employee",
  "description": "Role granted to workers on the payroll."
}' \
"http://localhost:8080/openidm/managed/role?_action=create"
{
  "_id": "d4f6b571-7e71-4901-8033-090a15098867",
  "_rev": "00000000ba0f5c8d",
  "name": "Employee",
  "description": "Role granted to workers on the payroll."
}
```

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "name": "Contractor",
  "description": "Role granted to contract workers."
}' \
"http://localhost:8080/openidm/managed/role?_action=create"
{
  "_id": "95899c38-e483-4d89-8aec-a88baab0603a",
  "_rev": "00000000c7cb5c0f",
  "name": "Contractor",
  "description": "Role granted to contract workers."
}
```

Note

Make sure to record these two role IDs, as they are required to complete the sample.

3. Reconcile the repository.

The `sync.json` configuration file for this sample includes two mappings: `systemLdapAccounts_managedUser`, which synchronizes users from the source LDAP server with the target IDM repository; and `managedUser_systemLdapAccounts`, which synchronizes changes from the repository with the LDAP server.

Run a reconciliation operation for the first mapping, either by using the Admin UI, or over the REST interface:

- To use the Admin UI, select Configure > Mapping, click on the first mapping (System/Ldap/Account --> Managed User) and click Reconcile.
- To use the REST interface, run the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemLdapAccounts_managedUser&waitForCompletion=true"
{
  "_id": "61abc9a3-a9cb-4d4b-b063-17891c3b355c-2541",
  "state": "SUCCESS"
}
```

The sample is now ready to demonstrate provisioning roles.

Run the Sample

This section assumes that you have reconciled the managed user repository to populate it with the users from the LDAP server, and that you have created the Employee and Contractor roles.

This part of the sample demonstrates the following features of the roles implementation:

- "Add Assignments to a Role Definition"
- "Grant a Role to a User and Observe that User's Role Assignments"
- "Propagate Assignments to an External System"
- "Remove a Role Grant From a User and Observe That User's Role Assignments"

Add Assignments to a Role Definition

An **assignment** is the logic that provisions a managed user to an external system, based on some criteria. The most common use case of an assignment is the provisioning of specific attributes to an external system, based on the role or roles that the managed user has been granted. Assignments are sometimes called *entitlements*.

In this section, you will create an assignment and add it to the Employee role that you created previously. This section assumes the following scenario:

example.com's policy requires that every employee has the correct value for their **employeeType** in their corporate directory (DS).

1. Display the roles that you created in the previous section:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/role?_queryFilter=true"
{
  "result": [
    {
      "_id": "d4f6b571-7e71-4901-8033-090a15098867",
      "_rev": "00000000ba0f5c8d",
      "name": "Employee",
      "description": "Role granted to workers on the payroll."
    },
    {
      "_id": "95899c38-e483-4d89-8aec-a88baab0603a",
      "_rev": "00000000c7cb5c0f",
      "name": "Contractor",
      "description": "Role granted to contract workers."
    }
  ],
  ...
}
```

Tip

Display the roles in the Admin UI by selecting Manage > Role.

2. Create a new managed assignment named Employee.

The assignment is specifically for the mapping from the managed user repository to the LDAP server. The assignment sets the value of the **employeeType** attribute on the LDAP server to **Employee**:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-type: application/json" \
```



```
--request POST \
--data '{
  "name" : "Employee",
  "description": "Assignment for employees.",
  "mapping" : "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "employeeType",
      "value": [
        "Employee"
      ],
      "assignmentOperation" : "mergeWithTarget",
      "unassignmentOperation" : "removeFromTarget"
    }
  ]
}' \
"http://localhost:8080/openidm/managed/assignment?_action=create"
{
  "_id": "1bbda95c-2a89-4e09-9719-8957849febeb",
  "_rev": "00000000ca15975d",
  "name": "Employee",
  "description": "Assignment for employees.",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "employeeType",
      "value": [
        "Employee"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ]
}
```

Tip

Create the assignment in the Admin UI:

1. Select Manage > Assignment, and click New Assignment.
2. Enter a name and description for the assignment, and select the mapping for which the assignment is applied (managedUser_systemLdapAccounts).
3. Click Add Assignment.
4. Select the Attributes tab, and click Add an Attribute.
5. Select employeeType, click item, select string, and enter the value Employee.
6. Click Save.

3. Add the assignment to the Employee role that you created previously.

Assignments are implemented as *relationship objects*. This means that you add an assignment to a role by *referencing* the assignment in the role's `assignments` field:

This command patches the Employee role to update its `assignments` field:

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
{
  "operation" : "add",
  "field" : "/assignments/-",
  "value" : { "_ref": "managed/assignment/1bbda95c-2a89-4e09-9719-8957849febeb"}
}]' \
"http://localhost:8080/openidm/managed/role/d4f6b571-7e71-4901-8033-090a15098867"
{
  "_id": "d4f6b571-7e71-4901-8033-090a15098867",
  "_rev": "00000000ba0f5c8d",
  "name": "Employee",
  "description": "Role granted to workers on the payroll."
}
```

Tip

Add the assignment to the role in the Admin UI:

1. Select Manage > Role, and select the Employee role.
2. On the Managed Assignments tab, click Add Managed Assignments.
3. Select the Employee assignment and click Add.

Grant a Role to a User and Observe that User's Role Assignments

When a role is granted to a user (by updating the users `roles` property), any assignments that are referenced by the role are automatically referenced in the user's `assignments` property.

In this section, we will grant the Employee role we created previously to the user Barbara Jensen, who was created in the managed/user repository during the reconciliation from DS.

1. Before you can update Barbara Jensen's entry, determine the identifier of her entry by querying her username, `bjensen`, and requesting only her `_id` field:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=/userName+eq+'bjensen'&_fields=_id"
{
  "result": [
    {
      "_id": "57356103-a1d5-4aaa-bcc5-a640147704e0",
      "_rev": "000000006688d203"
    }
  ],
  ...
}
```

From the output, observe that bjensen's `_id` is `57356103-a1d5-4aaa-bcc5-a640147704e0`.

2. Update bjensen's entry by adding a reference to the ID of the Employee role as a value of her `roles` attribute. Make sure to use the unique ID from your command output.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-type: application/json" \
--request PATCH \
--data '[
{
  "operation": "add",
  "field": "/roles/-",
  "value": { "_ref": "managed/role/d4f6b571-7e71-4901-8033-090a15098867" }
}]' \
"http://localhost:8080/openidm/managed/user/57356103-a1d5-4aaa-bcc5-a640147704e0"
{
  "_id": "57356103-a1d5-4aaa-bcc5-a640147704e0",
  "_rev": "000000005498d7b5",
  "displayName": "Barbara Jensen",
  "description": "Created for OpenIDM",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "telephoneNumber": "1-360-229-7105",
  "sn": "Jensen",
  "userName": "bjensen",
  "accountStatus": "active",
  "effectiveAssignments": [
    {
      "_rev": "00000000ca15975d",
      "_id": "1bbda95c-2a89-4e09-9719-8957849febeb",
      "name": "Employee",
      "description": "Assignment for employees.",
      "mapping": "managedUser_systemLdapAccounts",
      "attributes": [
        {
          "name": "employeeType",
          "value": [
```

```

    "Employee"
  ],
  "assignmentOperation": "mergeWithTarget",
  "unassignmentOperation": "removeFromTarget"
}
]
},
"effectiveRoles": [
{
  "_refResourceCollection": "managed/role",
  "_refResourceId": "d4f6b571-7e71-4901-8033-090a15098867",
  "_ref": "managed/role/d4f6b571-7e71-4901-8033-090a15098867"
}
]
}

```

Tip

Assign the role to bjensen by using the Admin UI:

1. Select Manage > User, and click on bjensen's entry.
2. On the Provisioning Roles tab, click Add Provisioning Roles.
3. Select the Employee role and click Add.

3. Take a closer look at bjensen's entry, specifically at her roles, effective roles and effective assignments:

```

curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=/userName+eq+'bjensen'&_fields=_id,userName,roles,effectiveRoles,effectiveAssignments"
{
  "result": [
    {
      "_id": "57356103-a1d5-4aaa-bcc5-a640147704e0",
      "_rev": "0000000005498d7b5",
      "userName": "bjensen",
      "effectiveAssignments": [
        {
          "_rev": "00000000ca15975d",
          "_id": "1bbda95c-2a89-4e09-9719-8957849febeb",
          "name": "Employee",
          "description": "Assignment for employees.",
          "mapping": "managedUser_systemLdapAccounts",
          "attributes": [
            {
              "name": "employeeType",
              "value": [
                "Employee"
              ]
            }
          ]
        }
      ]
    }
  ]
}

```

```

    ],
    "assignmentOperation": "mergeWithTarget",
    "unassignmentOperation": "removeFromTarget"
  }
}
],
},
],
"effectiveRoles": [
  {
    "_refResourceCollection": "managed/role",
    "_refResourceId": "d4f6b571-7e71-4901-8033-090a15098867",
    "_ref": "managed/role/d4f6b571-7e71-4901-8033-090a15098867"
  }
],
"roles": [
  {
    "_ref": "managed/role/d4f6b571-7e71-4901-8033-090a15098867",
    "_refResourceCollection": "managed/role",
    "_refResourceId": "d4f6b571-7e71-4901-8033-090a15098867",
    "_refProperties": {
      "_id": "75441276-4ede-4655-855c-e13ed4b47e8e",
      "_rev": "00000000ccc59e6c"
    }
  }
]
}
],
...
}

```

Note that bjensen now has the calculated property `effectiveAssignments`, which includes the set of assignments that pertains to any user with the Employee role. Currently the assignment lists the `employeeType` attribute.

In the next section, you will see how the assignment is used to set the value of the `employeeType` attribute in the LDAP server.

Propagate Assignments to an External System

This section provides a number of steps that show how effective assignments are propagated out to the external systems associated with their mappings.

1. Verify that bjensen's `employeeType` has been set correctly in DS.

Because implicit synchronization is enabled by default, any changes made to a managed user object are pushed out to all the external systems for which mappings are configured.

So, because bjensen has an effective assignment that sets an attribute in her LDAP entry, you should immediately see the resulting change in her LDAP entry.

To verify that her entry has changed, run an `ldapsearch` on her entry and check the value of her `employeeType` attribute:

```
/path/to/openssl/bin/ldapsearch \
--port 1636 \
--useSSL \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--hostname localhost \
--baseDN "dc=example,dc=com" \
--bindDN uid=admin \
--bindPassword password \
--searchScope sub \
"(uid=bjensen)" dn uid employeeType isMemberOf
dn: uid=bjensen,ou=People,dc=example,dc=com
employeeType: Employee
uid: bjensen
isMemberOf: cn=openidm2,ou=Groups,dc=example,dc=com
```

Note that bjensen's `employeeType` attribute is correctly set to `Employee`.

2. To observe how a managed user's roles can be used to provision group membership in an external directory, we add the groups that an Employee and a Contractor should have in the corporate directory (DS) as assignment attributes of the respective roles.

First, look at the current `assignments` of the Employee role again:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/role/d4f6b571-7e71-4901-8033-090a15098867?
fields=assignments,name"
{
  "_id": "d4f6b571-7e71-4901-8033-090a15098867",
  "_rev": "00000000ba0f5c8d",
  "name": "Employee",
  "assignments": [
    {
      "_ref": "managed/assignment/1bbda95c-2a89-4e09-9719-8957849febeb",
      "_refResourceCollection": "managed/assignment",
      "_refResourceId": "1bbda95c-2a89-4e09-9719-8957849febeb",
      "_refProperties": {
        "_id": "94cb5abd-5358-42d7-ab96-0e6808a157aa",
        "_rev": "00000000cf18a2b6"
      }
    }
  ]
}
```

To update the `groups` attribute in bjensen's LDAP entry, you do not need to create a *new* assignment. You simply need to add the attribute for LDAP groups to the Employee assignment (`1bbda95c-2a89-4e09-9719-8957849febeb`):

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
```

```
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
{
  "operation": "add",
  "field": "/attributes/-",
  "value": {
    "name": "ldapGroups",
    "value": [
      "cn=Employees,ou=Groups,dc=example,dc=com",
      "cn=Chat Users,ou=Groups,dc=example,dc=com"
    ],
    "assignmentOperation": "mergeWithTarget",
    "unassignmentOperation": "removeFromTarget"
  }
}
]' \
"http://localhost:8080/openidm/managed/assignment/1bbda95c-2a89-4e09-9719-8957849febeb"
{
  "_id": "1bbda95c-2a89-4e09-9719-8957849febeb",
  "_rev": "000000007248f2bc",
  "name": "Employee",
  "description": "Assignment for employees.",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "employeeType",
      "value": [
        "Employee"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    },
    {
      "name": "ldapGroups",
      "value": [
        "cn=Employees,ou=Groups,dc=example,dc=com",
        "cn=Chat Users,ou=Groups,dc=example,dc=com"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ]
}
```

So, the Employee assignment now sets two attributes on the LDAP system - the `employeeType` attribute, and the `ldapGroups` attribute.

Tip

To add more attributes to the Employee assignment in the Admin UI:

1. Select Manage > Assignment, and click on the Employee assignment.
2. On the Attributes tab, select Add an attribute and select the `ldapGroups` attribute.
3. Enter the values

```
cn=Employees,ou=Groups,dc=example,dc=com
```

and

```
cn=Chat Users,ou=Groups,dc=example,dc=com
```

and click Save.

3. With the implicit synchronization between the managed user repository and DS, bjensen should now be a member of the `cn=Employees` and `cn=Chat Users` groups in LDAP.

You can verify this with the following `ldapsearch` command. This command returns bjensen's group membership, in her `isMemberOf` attribute:

```
/path/to/openssl/bin/ldapsearch \
--port 1636 \
--useSSL \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--hostname localhost \
--baseDN "dc=example,dc=com" \
--bindDN uid=admin \
--bindPassword password \
--searchScope sub \
"(uid=bjensen)" dn uid employeeType isMemberOf
dn: uid=bjensen,ou=People,dc=example,dc=com
employeeType: Employee
uid: bjensen
isMemberOf: cn=Employees,ou=Groups,dc=example,dc=com
isMemberOf: cn=openidm2,ou=Groups,dc=example,dc=com
isMemberOf: cn=Chat Users,ou=Groups,dc=example,dc=com
```

You can also check bjensen's group membership by querying her object in the LDAP system, using the REST interface:


```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account?_queryFilter=/uid+sw
+ 'bjensen' & _fields=dn,uid,employeeType,ldapGroups"
{
  "result": [
    {
      "id": "887732e8-3db2-31bb-b329-20cd6fcec05",
      "dn": "uid=bjensen,ou=People,dc=example,dc=com",
      "uid": "bjensen",
      "employeeType": [
        "Employee"
      ],
      "ldapGroups": [
        "cn=Chat Users,ou=Groups,dc=example,dc=com",
        "cn=openidm2,ou=Groups,dc=example,dc=com",
        "cn=Employees,ou=Groups,dc=example,dc=com"
      ]
    }
  ],
  ...
}
```

In the original LDIF file, bjensen was already a member of the openidm2 group. You can ignore this group for the purposes of this sample.

Tip

Use the Admin UI to see bjensen's LDAP groups as follows:

1. Select Manage > User, and select bjensen's entry.
2. On the Linked Systems tab, scroll down to the ldapGroups item.

4. Now, create a new assignment that will apply to Contract employees, and add that assignment to the Contractor role.

Create the Contractor assignment with the following command. This assignment sets the value of the `employeeType` attribute to `Contractor`, and updates the user's `ldapGroups` attribute to include the `cn=Contractors` group:

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "name": "Contractor",
  "description": "Contractor assignment for contract workers.",
}
```

```

"mapping": "managedUser_systemLdapAccounts",
"attributes": [
  {
    "name": "ldapGroups",
    "value": [
      "cn=Contractors,ou=Groups,dc=example,dc=com"
    ],
    "assignmentOperation": "mergeWithTarget",
    "unassignmentOperation": "removeFromTarget"
  },
  {
    "name": "employeeType",
    "value": [
      "Contractor"
    ],
    "assignmentOperation": "mergeWithTarget",
    "unassignmentOperation": "removeFromTarget"
  }
]
} \
"http://localhost:8080/openidm/managed/assignment?action=create"
{
  "_id": "30323ed3-d885-4d09-94ca-c8f3b3408296",
  "_rev": "00000000db43da70",
  "name": "Contractor",
  "description": "Contractor assignment for contract workers.",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "ldapGroups",
      "value": [
        "cn=Contractors,ou=Groups,dc=example,dc=com"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    },
    {
      "name": "employeeType",
      "value": [
        "Contractor"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ]
}
}

```

Note the ID of the Contractor assignment (**30323ed3-d885-4d09-94ca-c8f3b3408296** in this example).

Tip

To create the assignment using the Admin UI, see "Add Assignments to a Role Definition".

5. Add the Contractor assignment to the Contractor role:

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
{
  "operation": "add",
  "field": "/assignments/-",
  "value": {
    "_ref": "managed/assignment/30323ed3-d885-4d09-94ca-c8f3b3408296"
  }
}
]' \
"http://localhost:8080/openidm/managed/role/95899c38-e483-4d89-8aec-a88baab0603a"
{
  "_id": "95899c38-e483-4d89-8aec-a88baab0603a",
  "_rev": "00000000c7cb5c0f",
  "name": "Contractor",
  "description": "Role granted to contract workers."
}
```

Tip

Add the Contractor assignment to the Contractor role in the Admin UI, as follows:

1. Select Manage > Role, and select the Contractor role.
2. On the Managed Assignments tab, click Add Managed Assignment.
3. Select the Contractor assignment from the dropdown list, and click Add.

6. Next, we need to grant the Contractor role to user jdoe. Before we can patch jdoe's entry, we need to know their system-generated ID. To obtain the ID, query jdoe's entry as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=/userName+eq+'jdoe'&_fields=_id"
{
  "result": [
    {
      "_id": "0d8a1d10-62e0-43aa-9892-ec51258771d0",
      "_rev": "000000005daacae0"
    }
  ],
  ...
}
```

For this example, you can see that jdoe's `_id` is `0d8a1d10-62e0-43aa-9892-ec51258771d0`.

- Update jdoe's entry by adding a reference to the ID of the Contractor role as a value of their **roles** attribute:

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
{
  "operation": "add",
  "field": "/roles/-",
  "value": {
    "_ref": "managed/role/95899c38-e483-4d89-8aec-a88baab0603a"
  }
}
]' \
"http://localhost:8080/openidm/managed/user/0d8a1d10-62e0-43aa-9892-ec51258771d0"
{
  "_id": "0d8a1d10-62e0-43aa-9892-ec51258771d0",
  "_rev": "00000000aa64591e",
  "displayName": "John Doe",
  "description": "Created for OpenIDM",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "1-415-599-1100",
  "sn": "Doe",
  "userName": "jdoe",
  "accountStatus": "active",
  "effectiveAssignments": [
    {
      "_rev": "00000000db43da70",
      "_id": "30323ed3-d885-4d09-94ca-c8f3b3408296",
      "name": "Contractor",
      "description": "Contractor assignment for contract workers.",
      "mapping": "managedUser_systemLdapAccounts",
      "attributes": [
        {
          "name": "ldapGroups",
          "value": [
            "cn=Contractors,ou=Groups,dc=example,dc=com"
          ],
          "assignmentOperation": "mergeWithTarget",
          "unassignmentOperation": "removeFromTarget"
        },
        {
          "name": "employeeType",
          "value": [
            "Contractor"
          ],
          "assignmentOperation": "mergeWithTarget",
          "unassignmentOperation": "removeFromTarget"
        }
      ]
    }
  ],
  "effectiveRoles": [
    {
```

```

    "_refResourceCollection": "managed/role",
    "_refResourceId": "95899c38-e483-4d89-8aec-a88baab0603a",
    "_ref": "managed/role/95899c38-e483-4d89-8aec-a88baab0603a"
  }
}
}

```

Tip

Grant the Contractor role to jdoe by using the Admin UI, as follows:

1. Select Manage > User, and click on jdoe's entry.
2. On the Provisioning Roles tab, click Add Provisioning Roles.
3. Select the Contractor role and click Add.

8. Check jdoe's entry on the LDAP system.

With the implicit synchronization between the managed user repository and DS, jdoe should now be a member of the **cn=Contractors** group in LDAP. In addition, his **employeeType** should have been set to **Contractor**.

You can verify this with the following REST query. This command returns jdoe's group membership, in his **isMemberOf** attribute, and his **employeeType**:

```

curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account?_queryFilter=/uid+sw
+'jdoe'&_fields=dn,uid,employeeType,ldapGroups"
{
  "result": [
    {
      "_id": "0da50512-79bb-3461-bd04-241ee4c785bf",
      "dn": "uid=jdoe,ou=People,dc=example,dc=com",
      "uid": "jdoe",
      "employeeType": [
        "Contractor"
      ],
      "ldapGroups": [
        "cn=openidm,ou=Groups,dc=example,dc=com",
        "cn=Contractors,ou=Groups,dc=example,dc=com"
      ]
    }
  ],
  ...
}

```

Tip

Use the Admin UI to see jdoo's LDAP groups as follows:

1. Select Manage > User, and select jdoo's entry.
2. On the Linked Systems tab, scroll down to the ldapGroups item.

Note

When working with large groups in LDAP services such as DS, you should use dynamic groups instead of static groups. The steps laid out above for setting assignments and roles work with the exception of how you add a user to a group: in dynamic groups, membership is determined by whether a user has an attribute the group is configured to search for.

For example, if the Employees group was a dynamic group, membership might be set based on the `employeeType` attribute directly, by setting the `memberURL` in the group to `ldap:///ou=People,dc=example,dc=com??sub?(employeeType=Employee)`. You would then remove the `ldapGroups` attribute from the Employee assignment, since group membership is handled by `employeeType`.

This membership won't be listed in the `ldapGroups` attribute in IDM (since it is no longer set there), but can be verified by querying DS directly:

```
/path/to/openssl/bin/ldapsrch \
--port 1636 \
--useSSL \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--hostname localhost \
--baseDN "dc=example,dc=com" \
--bindDN uid=admin \
--bindPassword password \
--searchScope sub \
"(uid=bjensen)" dn uid employeeType isMemberOf
dn: uid=bjensen,ou=People,dc=example,dc=com
employeeType: Employee
uid: bjensen
isMemberOf: cn=Employees,ou=Groups,dc=example,dc=com
isMemberOf: cn=openidm2,ou=Groups,dc=example,dc=com
isMemberOf: cn=Chat Users,ou=Groups,dc=example,dc=com
```

For more information about dynamic groups in DS, see *Dynamic Groups* in the *Configuration Guide* for ForgeRock Directory Services.

Remove a Role Grant From a User and Observe That User's Role Assignments

In this section, you will remove the Contractor role from jdoo's managed user entry and observe the subsequent change to jdoo's managed assignments, and to the corresponding attributes in DS.

1. Before you change jdoo's roles, view his entry again to examine his current roles:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=/userName+eq+'jdoe'&_fields=_id,roles"
{
  "result": [
    {
      "_id": "0d8a1d10-62e0-43aa-9892-ec51258771d0",
      "_rev": "00000000aa64591e",
      "roles": [
        {
          "_ref": "managed/role/95899c38-e483-4d89-8aec-a88baab0603a",
          "_refResourceCollection": "managed/role",
          "_refResourceId": "95899c38-e483-4d89-8aec-a88baab0603a",
          "_refProperties": {
            "_id": "de68f5d9-baf9-49ca-8db5-96f0a382946d",
            "_rev": "00000000e299a021"
          }
        }
      ]
    }
  ],
  ...
}
```

Note the following IDs in this example output:

1. The ID of jdoe's user object is `0d8a1d10-62e0-43aa-9892-ec51258771d0`.
2. The ID of the contractor role (`_refResourceId`) is `95899c38-e483-4d89-8aec-a88baab0603a`.
3. The ID of the *relationship* that expresses the role grant is `de68f5d9-baf9-49ca-8db5-96f0a382946d`.

You will need these IDs in the next step.

Tip

View jdoe's current roles in the Admin UI:

1. Select Manage > User, and select jdoe's entry.
2. The Provisioning Roles tab lists the current roles.

2. Remove the Contractor role from jdoe's entry by sending a DELETE request to his user entry, specifying the *relationship ID*:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/managed/user/0d8a1d10-62e0-43aa-9892-ec51258771d0/roles/de68f5d9-baf9-49ca-8db5-96f0a382946d"
{
  "_id": "de68f5d9-baf9-49ca-8db5-96f0a382946d",
  "_rev": "00000000e299a021",
  "_ref": "managed/role/95899c38-e483-4d89-8aec-a88baab0603a",
  "_refResourceCollection": "managed/role",
  "_refResourceId": "95899c38-e483-4d89-8aec-a88baab0603a",
  "_refProperties": {
    "_id": "de68f5d9-baf9-49ca-8db5-96f0a382946d",
    "_rev": "00000000e299a021"
  }
}
```

The output shows that the *relationship* between the user and the role was deleted.

Tip

Use the Admin UI to remove the Contractor role from jdoe's entry as follows:

1. Select Manage > User, and select jdoe's entry.
2. On the Provisioning Roles tab, check the box next to the Contractor role and click Remove Selected Provisioning Roles.

3. Verify jdoe's `employeeType` and `ldapGroups`.

The removal of the Contractor role causes a synchronization operation to be run on jdoe's entry. His `employeeType` and `ldapGroups` attributes in DS should be reset to what they were before he was granted the Contractor role.

Check jdoe's attributes by querying his object in the LDAP directory, over the REST interface:


```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account?_queryFilter=/uid+sw
+jdoe'&_fields=dn,uid,employeeType,ldapGroups"
{
  "result": [
    {
      "_id": "0da50512-79bb-3461-bd04-241ee4c785bf",
      "dn": "uid=jdoe,ou=People,dc=example,dc=com",
      "uid": "jdoe",
      "employeeType": [],
      "ldapGroups": [
        "cn=openidm,ou=Groups,dc=example,dc=com"
      ]
    }
  ],
  ...
}
```

Tip

Use the Admin UI to see jdoe's LDAP groups as follows:

1. Select Manage > User, and select jdoe's entry.
2. On the Linked Systems tab, scroll down to the ldapGroups item.

This concludes the provisioning with roles sample. For more information about roles, assignments, and how to manipulate them, see "Managed Roles" in the *Object Modeling Guide*.

Chapter 18

Provision Users With Workflow

This sample demonstrates a typical workflow use case — provisioning new users.

The sample uses the Admin UI to set up the initial users and roles, then shows how users can complete their registration process in the End User UI.

The sample simulates the following scenario:

- An existing employee requests that an outside contractor be granted access to an organization's system.
- The *system* in this case, is the IDM managed user repository and a remote HR data source, represented by a CSV file (`hr.csv`).
- User roles are stored separately, in a second CSV file (`roles.csv`).

The sample has three mappings—two for the bidirectional synchronization of the managed user repository and the HR data store, and one for the synchronization of the roles data to the managed repository.

Prepare the Sample

In this section, you start IDM, configure the outbound email service, and reconcile user and role data. The reconciliation operations create two managed users, `user1` and `manager1`, and two managed roles, `employee` (assigned to `user1`) and `manager` (assigned to `manager1`).

Important

Workflows are not supported with a DS repository. Before you test this sample, install a JDBC repository.

1. Edit the `/path/to/openidm/samples/provisioning-with-workflow/conf/datasource.jdbc-default.json` file with the details of your JDBC repository. For more information, see "Select a Repository" in the *Installation Guide*.
2. Start IDM with the configuration for the provisioning sample:

```
/path/to/openidm/startup.sh -p samples/provisioning-with-workflow
```
3. Log in to the Admin UI in the *Setup Guide*.

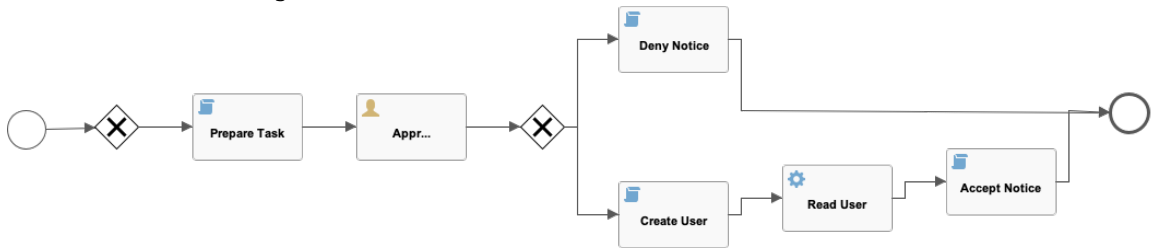
4. Configure the outbound email service:
 - a. From the navigation bar, click Configure > Email Settings.
 - b. On the Email Settings page, enable the outbound mail service, enter the connection information, and click Save.
5. Enable Password Reset.
 - a. From the navigation bar, click Configure > Password Reset.
 - b. On the Password Reset page, enable password reset, enter the applicable information, and click Save.

Note

For additional password reset information, see "Configuring Emails for Password Reset" in the *Self-Service Reference*.

6. Reconcile the role and user data.
 - a. From the navigation bar, click Configure > Mappings.
 - b. Select the first mapping (`systemRolesFileRole_managedRole`) and click Reconcile.
 - c. To verify the reconciliation, from the navigation bar, click Manage > Role.
IDM displays the two roles created in the previous step: `employee` and `manager`.
 - d. From the navigation bar, click Configure > Mappings.
 - e. Select the second mapping (`systemCsvfileAccounts_managedUser`), and click Reconcile.
The reconciliation operation creates the top-level managers (users who do not have their own `manager` property) in the managed user repository. In this sample, there is only one top-level manager (`manager1`).
 - f. Select the second mapping again (`systemCsvfileAccounts_managedUser`), and click Reconcile.
This reconciliation operation creates the employees of the managers that were created by the previous reconciliation. In this sample, there is only one employee (`employee1`).
 - g. From the navigation bar, click Manage > User, and verify the users `manager1` and `user1` exist.
7. Verify the relationships between the new user and role objects:
 - a. Click `user1`. The Manager field displays `manager1` for this user.
 - b. Click the Authorization Roles tab. `user1` has two roles: `openidm-authorized` and `employee`.

- c. From the breadcrumb link at the top of the page, click User, and select **manager1**. The Manager field is empty for this user.
 - d. Click the Authorization Roles tab. **manager1** has three roles: **manager**, **openidm-authorized**, and **openidm-tasks-manager**.
8. Verify the available workflows.
- a. From the navigation bar, click Manage > Processes.
 - b. On the Workflow Processes page, select the Definitions tab.
 - c. From the Definitions list, click Contractor onboarding process. IDM displays a diagram similar to the following:




9. Log out of the Admin UI.

Run the Sample

During this part of the sample, an existing employee initiates a *Contractor Onboarding* process. This process is a request to add a contractor to the managed user repository, with an option to include the contractor in the original HR data source (**hr.csv**).

When the employee has completed the required form, the request is sent to the manager for approval. Any user with the role **manager** can claim the approval task. If the request is approved, an email is sent to the address provided in the initial form, with a request for the contractor to reset their password. When the password reset has been completed, the contractor is created in the managed user repository. If a request was made to add the contractor to the original HR data source, this is done when the manager approves the request.

1. Log in to the End User UI (<https://localhost:8443/>) as the user you created in the previous section (**user1**) with password **Welcome1**.
2. Navigate to the dashboard, with the Dashboard icon () or the Menu icon () and select Dashboard.
3. Initiate the provisioning workflow as user1:



- a. Scroll down to the Start a Process menu, and click Start next to the Contractor onboarding process.

Complete the form for the sample user you will be creating. Use an email address to which you have access. You'll need access to complete this workflow.

Activate the Create in CSV File option, which enables implicit synchronization from the managed user repository to the `hr.csv` file.

Note

Note that `user1` does not provide a password for this user. A password reset request is sent to the email address provided on this form to ensure that only the actual contractor can log in with this account.

- b. Select Submit to initiate the process.
 - c. Log out of the End User UI.
4. Approve the workflow task as `manager1`:
- a. Log in to the End User UI as `manager1`, with password `Welcome1`.
 - b. Navigate to the dashboard, with the Dashboard icon (). Alternatively, select the Menu icon () and select Dashboard.
 - c. Under Unassigned Tasks, locate the Approve Contractor task, select Assign, and select Assign to Me.
 - d. Approve Contractor is now listed under My Tasks.
Select Edit next to the task name.
 - e. Review the form content. (It is the same content that you provided as `user1`.)
Assuming the information is acceptable, select Accept.
 - f. Log out of the End User UI.
5. Verify that the contractor has been created in the HR data source (`hr.csv`):

```
more /path/to/openidm/samples/provisioning-with-workflow/data/hr.csv
```

```
"username","firstname","lastname","manager","department","jobTitle",    ...,"password",...
"user1",  "Ordinary", "Employee","manager1","depl",    "job1",    ...,"Welcome1",...
"manager1","Big",    "Manager", "",    "depl",    "Manager",    ...,"Welcome1",...
"bjensen", "Barbara", "Jensen", "user1", "Payroll", "Payroll clerk",...,    ....
```

Note the addition of the new contractor entry, in this case, bjensen. Note also that there is no value for the `password` field, and that `user1` is the manager of the new contractor.

6. Complete the password reset process:

- a. Verify the inbox of the email account that you provided when you completed the initial form.

You should have received two emails — one with the subject "Your account has been created" and one with the subject "Reset your password".

- b. Open the password reset email and select `Password reset link`.

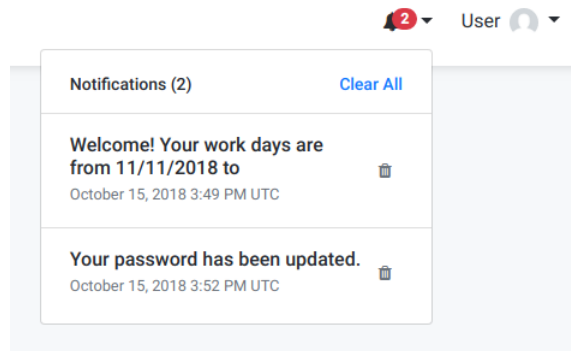
The link takes you to the End User UI, with the option to Reset Your Password.

- c. Enter a new password, and select Change Password.

The password that you enter here must comply with the password policy that is configured for managed users. For more information, see "Enforcing Password Policy" in the *Security Guide*.

- d. Select Sign In and connect with the username and newly created password.
- e. Select the notifications icon (🔔), and you should see welcome and password update messages:

Notifications for new users



7. Verify that the new password has been propagated to the HR data source:

Open `/path/to/openidm/samples/provisioning-with-workflow/data/hr.csv` and note that the password for the contractor has been added to their entry.

If you declined the approval request, the user is not created in either the managed user repository, or in the HR CSV file.

Chapter 19

Connect to DS With ScriptedREST

This sample uses the scripted REST connector to interact with the ForgeRock Directory Services (DS) REST API, using Groovy scripts. The sample demonstrates reconciliation, implicit sync, and liveSync between the IDM repository and a DS instance.

The scripted REST connector is bundled with IDM in the JAR `openidm/connectors/scriptedrest-connector-1.5.20.11.jar`.

The Groovy scripts required for the sample are located in the `samples/scripted-rest-with-dj/tools` directory. You must customize these scripts to address the requirements of your specific deployment, however, the sample scripts are a good starting point on which to base your customization.

Important

The Rest2ldap HTTP endpoint provided with DS is an evolving interface. As such, compatibility between versions is not guaranteed. This sample was tested with DS 7.

Set Up DS

1. Set up DS, but remove the line `--set ds-user-data/ldifFile:Example.ldif`.
2. Optionally, you can enable an HTTP access logger on the DS server.
3. Import the data required for the sample:

```
/path/to/openssl/bin/openssl \
--port 1636 \
--useSSL \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--hostname localhost \
--bindDN uid=admin \
--bindPassword password \
--filename /path/to/openidm/samples/scripted-rest-with-dj/data/ldap.ldif
# ADD operation successful for DN dc=example,dc=com

# ADD operation successful for DN ou=Administrators,dc=example,dc=com

# ADD operation successful for DN uid=idm,ou=Administrators,dc=example,dc=com

# ADD operation successful for DN ou=People,dc=example,dc=com

# ADD operation successful for DN ou=Groups,dc=example,dc=com
```

- Set up the access control instructions (ACIs) that enable the IDM administrator user to read the DS external change log:

```
/path/to/openssl/bin/dsconfig set-access-control-handler-prop \
--port 4444 \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--hostname localhost \
--bindDN uid=admin \
--bindPassword password \
--add global-aci:"(target=\"ldap:///cn=changelog\")(targetattr=\"*|+|\") \
(version 3.0; acl \"IDM can access cn=changelog\"; \
allow (read,search,compare) \
userdn=\"ldap:///uid=idm,ou=Administrators,dc=example,dc=com\");\" \
--no-prompt
```

```
/path/to/openssl/bin/dsconfig set-access-control-handler-prop \
--port 4444 \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--hostname localhost \
--bindDN uid=admin \
--bindPassword password \
--add global-aci:"(targetcontrol=\"1.3.6.1.4.1.26027.1.5.4\") \
(version 3.0; acl \"IDM changelog control access\"; \
allow (read) \
userdn=\"ldap:///uid=idm,ou=Administrators,dc=example,dc=com\");\" \
--no-prompt
```

- Enable the default Rest2ldap HTTP endpoint:

```
/path/to/openssl/bin/dsconfig set-http-endpoint-prop \
--port 4444 \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--hostname localhost \
--bindDN uid=admin \
--bindPassword password \
--endpoint-name /api \
--set authorization-mechanism:"HTTP Basic" \
--set config-directory:config/rest2ldap/endpoints/api \
--set enabled:true \
--no-prompt
```

For more information, see [HTTP User APIs](#) in the *DS Configuration Guide*.

- Replace the default DS REST to LDAP configuration with the configuration for this sample:

```
cp /path/to/openidm/samples/scripted-rest-with-dj/data/example-v1.json /path/to/openssl/config/
rest2ldap/endpoints/api/
```

- Restart DS for the configuration change to take effect.

```
/path/to/openssl/bin/stop-ds --restart
Stopping Server...
...
The Directory Server has started successfully
```


Run the Sample

This section illustrates the basic CRUD operations on users and groups using the ScriptedREST connector and the DS REST API. Note that the power of the Groovy connector is in the associated Groovy scripts, and their application in your particular deployment. The scripts provided with this sample are specific to the sample and customization of the scripts is required.

1. Start IDM with the configuration for the ScriptedREST sample:

```
cd /path/to/openidm/  
./startup.sh -p samples/scripted-rest-with-dj
```

2. Check that the scripted REST connector can reach the DS instance by obtaining the connector status over REST:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/scriptedrest?_action=test"
{
  "name": "scriptedrest",
  "enabled": true,
  "config": "config/provisioner.openicf/scriptedrest",
  "connectorRef": {
    "bundleVersion": "[1.5.0.0,1.6.0.0)",
    "bundleName": "org.forgerock.openicf.connectors.scriptedrest-connector",
    "connectorName": "org.forgerock.openicf.connectors.scriptedrest.ScriptedRESTConnector"
  },
  "displayName": "Scripted REST Connector",
  "objectTypes": [
    "__ALL__",
    "account",
    "group"
  ],
  "ok": true
}
```

3. Create a group entry on the DS server:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "cn": "group1"
}' \
"http://localhost:8080/openidm/system/scriptedrest/group?_action=create"
{
  "_id": "group1",
  "members": null,
  "created": "2020-07-21T23:04:25Z",
  "cn": "group1",
  "displayName": "group1"
}
```

4. Create a user entry on the DS server. This command creates a user with **uid** scarter:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "givenName": "Steven",
  "familyName": "Carter",
  "emailAddress": "scarter@example.com",
  "telephoneNumber": "444-444-4444",
  "password": "5up35tr0ng",
  "displayName": "Steven.Carter",
  "uid": "scarter"
}' \
"http://localhost:8080/openidm/system/scriptedrest/account?_action=create"
{
  "_id": "scarter",
  "givenName": "Steven",
  "groups": null,
  "displayName": "Steven.Carter",
  "emailAddress": "scarter@example.com",
  "uid": "scarter",
  "created": "2020-07-21T23:07:13Z",
  "familyName": "Carter",
  "telephoneNumber": "444-444-4444"
}
```

Notice that the user is not a member of any group.

5. Reconcile the DS server with the managed user repository:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemRestLdapUser_managedUser&waitForCompletion=true"
{
  "_id": "ee7534bd-ccfd-4f6a-bdc3-49caa6d2043c-547",
  "state": "SUCCESS"
}
```

6. The reconciliation creates a managed user with a server-assigned ID. To retrieve the ID, run the following query:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true&_fields=_id"
{
  "result": [
    {
      "_id": "4657a420-6608-410e-baa7-f64668cc500c",
      "_rev": "000000007995f006"
    }
  ],
  ...
}
```

7. To initialize liveSync set the sync token by running one liveSync operation over REST:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/scriptedrest/account?_action=liveSync"
{
  "connectorData": {
    "nativeType": "string",
    "syncToken": "8"
  }
}
```

8. Update Steven Carter's managed user entry, by modifying his telephone number. Specify the user ID that you retrieved previously:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
{
  "operation": "replace",
  "field": "telephoneNumber",
  "value": "555-555-5555"
}
]' \
"http://localhost:8080/openidm/managed/user/4657a420-6608-410e-baa7-f64668cc500c"
{
  "_id": "4657a420-6608-410e-baa7-f64668cc500c",
  "_rev": "0000000096edf021",
  "userName": "scarter",
  "mail": "scarter@example.com",
  "displayName": "Steven.Carter",
  "telephoneNumber": "555-555-5555",
  "givenName": "Steven",
  "sn": "Carter",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

9. The implicit synchronization mechanism between the managed user repository and DS propagates the change to DS. You can check this change by reading scarter's user entry in DS and noting the changed **telephoneNumber**:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/scriptedrest/account/scarter"
{
  "_id": "scarter",
  "familyName": "Carter",
  "givenName": "Steven",
  "created": "2018-02-07T10:14:31Z",
  "uid": "scarter",
  "groups": null,
  "emailAddress": "scarter@example.com",
  "displayName": "Steven.Carter",
  "telephoneNumber": "555-555-5555"
}
```

10. Now, update Steven Carter's entry on the DS server, by modifying his **givenName**:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
{
  "operation": "replace",
  "field": "givenName",
  "value": "Steve"
}
]' \
"http://localhost:8080/openidm/system/scriptedrest/account/scarter"
{
  "_id": "scarter",
  "givenName": "Steve",
  "groups": null,
  "displayName": "Steven.Carter",
  "emailAddress": "scarter@example.com",
  "uid": "scarter",
  "created": "2020-07-21T23:07:13Z",
  "familyName": "Carter",
  "telephoneNumber": "555-555-5555"
}
```

11. To propagate the change made on DS back to the managed user entry, launch a liveSync operation, either by defining a schedule, or directly over REST. The following command launches liveSync over REST:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/scriptedrest/account?_action=liveSync"
{
  "connectorData": {
    "nativeType": "string",
    "syncToken": "9"
  },
  "_rev": "000000000a585336",
  "_id": "SYSTEMSCRIPTEDRESTACCOUNT"
}
```

12. Verify that the changes were propagated by reading scarter's managed user entry and noting the changed **givenName**:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/4657a420-6608-410e-baa7-f64668cc500c"
{
  "_id": "4657a420-6608-410e-baa7-f64668cc500c",
  "_rev": "000000007937efb7",
  "userName": "scarter",
  "mail": "scarter@example.com",
  "displayName": "Steven.Carter",
  "telephoneNumber": "555-555-5555",
  "givenName": "Steve",
  "sn": "Carter",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

13. Add user scarter to the group you created previously, by updating the group entry:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--header "If-Match: *" \
--request PUT \
--data '{
  "_id": "group1",
  "members": [{"_id": "scarter"}]
}' \
"http://localhost:8080/openidm/system/scriptedrest/group/group1"
{
  "_id": "group1",
  "displayName": "group1",
  "created": "2018-02-07T10:14:12Z",
  "members": [
    {
      "_id": "scarter",
      "displayName": "Steven.Carter"
    }
  ],
  "cn": "group1",
  "lastModified": "2018-02-07T10:20:22Z"
}
```

14. Read Steven Carter's user entry in DS, to verify that he is now a member of group1:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/scriptedrest/account/scarter"
{
  "_id": "scarter",
  "givenName": "Steve",
  "groups": [
    {
      "_id": "group1"
    }
  ],
  "displayName": "Steven.Carter",
  "emailAddress": "scarter@example.com",
  "uid": "scarter",
  "created": "2020-07-21T23:07:13Z",
  "familyName": "Carter",
  "telephoneNumber": "555-555-5555"
}
```

15. Read the group entry to verify its members:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/scriptedrest/group/group1"
{
  "_id": "group1",
  "lastModified": "2020-07-21T23:17:09Z",
  "members": [
    {
      "_id": "scarter",
      "displayName": "Steven.Carter"
    }
  ],
  "created": "2020-07-21T23:04:25Z",
  "cn": "group1",
  "displayName": "group1"
}
```

16. Reconcile the DS groups with the managed group repository:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemRestLdapGroup_managedGroup&waitForCompletion=true"
{
  "_id": "ee7534bd-ccfd-4f6a-bdc3-49caa6d2043c-1477",
  "state": "SUCCESS"
}
```

17. The reconciliation creates a managed group with a server-assigned ID. To retrieve the group ID, run the following query:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request GET \
"http://localhost:8080/openidm/managed/group?_queryId=query-all-ids"
{
  "result": [
    {
      "_id": "67b5ec50-d5a6-4bfa-bb19-17965447ad00",
      "_rev": "00000000b0e95e9b"
    }
  ],
  ...
}
```

18. Read the managed group to verify that the DS group has been added, and that its members have been reconciled to the managed group repository. Specify the ID that you retrieved in the previous step:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/group/67b5ec50-d5a6-4bfa-bb19-17965447ad00"
{
  "_id": "67b5ec50-d5a6-4bfa-bb19-17965447ad00",
  "_rev": "00000000b0e95e9b",
  "members": [
    {
      "_id": "scarter",
      "displayName": "Steven.Carter"
    }
  ],
  "displayName": "group1"
}
```

19. Delete the DS user and group entries, returning the DS server to its initial state.


```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/system/scriptedrest/account/scarter"
{
  "_id": "scarter",
  "givenName": "Steve",
  "groups": [
    {
      "_id": "group1"
    }
  ],
  "displayName": "Steven.Carter",
  "emailAddress": "scarter@example.com",
  "uid": "scarter",
  "created": "2020-07-21T23:07:13Z",
  "familyName": "Carter",
  "telephoneNumber": "555-555-5555"
}
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/system/scriptedrest/group/group1"
{
  "_id": "group1",
  "lastModified": "2020-07-21T23:17:09Z",
  "members": null,
  "created": "2020-07-21T23:04:25Z",
  "cn": "group1",
  "displayName": "group1"
}
```

Chapter 20

Connect to Active Directory With the PowerShell Connector

This sample shows an implementation of the PowerShell Connector toolkit and provides a number of PowerShell scripts that let you perform basic CRUD (create, read, update, delete) operations on an Active Directory server.

The sample uses the MS Active Directory PowerShell module. For more information on this module, see the corresponding [Microsoft documentation](#).

Sample Overview

The generic *PowerShell Connector Toolkit* enables you to run PowerShell scripts on any external resource. The PowerShell Connector Toolkit is not a complete connector, in the traditional sense. Rather, it is a framework within which you must write your own PowerShell scripts to address the requirements of your Microsoft Windows ecosystem. You can use the PowerShell Connector Toolkit to create connectors that can provision any Microsoft system.

The PowerShell Connector Toolkit is available from the [ForgeRock BackStage](#) download site.

This sample assumes that IDM is running on a Windows system on the localhost. It also assumes that Active Directory and the ICF .NET connector server run on a remote Windows server. The PowerShell connector runs on the .NET connector server.

To use this sample for IDM instances installed on UNIX systems, adjust the relevant commands shown with PowerShell prompts.

Note

By default, the `Get-ADUser` and `Get-ADGroup` cmdlets are not thread safe. To avoid thread issues when you use this connector with Active Directory, you must set the pooling configuration properties as follows:

```
"UseInterpretersPool" : true,  
"MinInterpretersPoolSize" : 1,  
"MaxInterpretersPoolSize" : 10
```

For more information about these properties see "Configuring the PowerShell Connector" in the *Connectors Guide*.

Prepare the Sample

Run the commands in this procedure from the PowerShell command line. The continuation character used in the command is the back-tick (`).

1. Install, configure, and start the .NET connector server on the machine that is running an Active Directory Domain Controller or on a workstation on which the Microsoft Active Directory PowerShell module is installed.

For instructions on installing the .NET connector server, see "Set Up a .NET Connector Server" in the *Connectors Guide*.

2. Configure IDM to connect to the .NET connector server.
3. Download the PowerShell Connector Toolkit archive ([mspowershell-connector-1.4.7.0.zip](#)) from the ForgeRock BackStage download site.

Extract the archive and move the `MsPowerShell.Connector.dll` to the folder in which the connector server application (`connectorserver.exe`) is located.

4. Copy the PowerShell scripts and the ADSISearch module from the `samples\scripted-powershell-with-ad\tools` directory, to the machine on which the connector server is installed.

```
dir \path\to\openidm\samples\scripted-powershell-with-ad\tools
Directory: C:\path\to\openidm\samples\scripted-powershell-with-ad\tools

Mode                LastWriteTime         Length Name
----                -
-a----            4/3/2018   3:26 AM           4279 ADAAuthenticate.ps1
-a----            4/3/2018   3:26 AM           9055 ADCreate.ps1
-a----            4/3/2018   3:26 AM           3717 ADDelete.ps1
-a----            4/3/2018   3:26 AM          10756 ADSchema.ps1
-a----            4/3/2018   3:26 AM           4625 ADSearch.ps1
-a----            4/3/2018   3:26 AM           8064 ADSISearch.psm1
-a----            4/3/2018   3:26 AM           5918 ADSync.ps1
-a----            4/3/2018   3:26 AM           2408 ADTest.ps1
-a----            4/3/2018   3:26 AM          18406 ADUpdate.ps1
PS C:\path\to\openidm>
```

5. Copy the sample connector configuration for the PowerShell connector to your project's `conf` directory.

```
cp \path\to\openidm\samples\example-configurations\provisioners\provisioner.openicf-adpowershell.json
\path\to\openidm\conf
```

The following excerpt of the sample connector configuration shows the configuration properties:

```
"configurationProperties" : {
  "AuthenticateScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/
ADAuthenticate.ps1",
  "CreateScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADCreate.ps1",
  "DeleteScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADDelete.ps1",
  "SchemaScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADSchema.ps1",
  "SearchScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADSearch.ps1",
  "SyncScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADSync.ps1",
  "TestScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADTest.ps1",
  "UpdateScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADUpdate.ps1",
  "VariablesPrefix" : "Connector",
  "QueryFilterType" : "Ldap",
  "ReloadScriptOnExecution" : true,
  "UseInterpretersPool" : true,
  "SubstituteUidAndNameInQueryFilter" : true,
  "UidAttributeName" : "ObjectGUID",
  "NameAttributeName" : "DistinguishedName",
  "PsModulesToImport" : [
    "ActiveDirectory",
    "C:/openidm/samples/scripted-powershell-with-ad/tools/ADSIsearch.psml"
  ],
  "Host" : "",
  "Port" : null,
  "Login" : "",
  "Password" : null,
  "CustomProperties" : ["baseContext = CN=Users,DC=example,DC=com" ],
  "MinInterpretersPoolSize" : 1,
  "MaxInterpretersPoolSize" : 10
},
```

The sample connector configuration assumes that the scripts are located in `C:/openidm/samples/scripted-powershell-with-ad/tools/`. If you copied your scripts to a different location, or are using a different base context for search and synchronization operations such as `DC=example,DC=org`, adjust your connector configuration file accordingly.

Note that the ICF framework requires the path to use forward slash characters and not the backslash characters that you would expect in a Windows path.

The host, port, login and password of the machine on which Active Directory runs do not need to be specified here. By default the Active Directory cmdlets pick up the first available Domain Controller. In addition, the scripts are executed using the credentials of the .Net connector server.

Note

The `ReloadScriptOnExecution` property is set to `true` in this sample configuration. This setting causes script files to be reloaded each time the script is invoked. Having script files reloaded each time is suitable for

debugging purposes. However, this property should be set to `false` in production environments, as the script reloading can have a negative performance impact.

In addition, make sure that the value of the `connectorHostRef` property in the connector configuration file matches the value that you specified in the remote connector configuration file, in step 2 of this procedure. For example:

```
"connectorHostRef" : "dotnet",
```

Run the Sample

Because you have copied all of the required configuration files into the default IDM project, you can start IDM with the default configuration (that is, without the `-p` option):

```
\path\to\openidm\startup.bat
```

When IDM has started, test the sample by using the `curl` command-line utility. The following examples test the scripts that were provided in the `tools` directory.

1. Test the connector configuration, and whether IDM is able to connect to the .NET connector server with the following request:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0" `
--request POST `
"http://localhost:8080/openidm/system?_action=test"
[
{
  "ok": true,
  "connectorRef": {
    "bundleVersion": "[1.4.2.0,1.5.0.0)",
    "bundleName": "MsPowerShell.Connector",
    "connectorName": "Org.ForgeRock.OpenICF.Connectors.MsPowerShell.MsPowerShellConnector"
  },
  "objectTypes": [
    "ALL",
    "group",
    "account"
  ],
  "config": "config/provisioner.openicf/adpowershell",
  "enabled": true,
  "name": "adpowershell"
}
]
```

2. Query the users in your Active Directory with the following request:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0" `
--request GET `
"http://localhost:8080/openidm/system/adpowershell/account?_queryId=query-all-ids"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 1257,
  "result": [
    {
      "_id": "7c41496a-9898-4074-a537-bed696b6be92"
    },
    {
      "_id": "f2e08a5c-473f-4798-a2d5-d5cc27c862a9"
    },
    {
      "_id": "6feef4a0-b121-43dc-be68-a96703a49aba"
    },
    ...
  ]
}
```

3. To return the complete record of a specific user, include the ID of the user in the URL. The following request returns the record for Steven Carter:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0" `
--request GET `
"http://localhost:8080/openidm/system/adpowershell/account/6feef4a0-b121-43dc-be68-a96703a49aba"
{
  "_id": "6feef4a0-b121-43dc-be68-a96703a49aba",
  "postalCode": null,
  "passwordNotRequired": false,
  "cn": "Steven Carter",
  "name": "Steven Carter",
  "trustedForDelegation": false,
  "uSNChanged": "47219",
  "manager": null,
  "objectGUID": "6feef4a0-b121-43dc-be68-a96703a49aba",
  "modifyTimeStamp": "11/27/2014 3:37:16 PM",
  "employeeNumber": null,
  "sn": "Carter",
  "userAccountControl": 512,
  "passwordNeverExpires": false,
  "displayName": "Steven Carter",
  "initials": null,
  "pwdLastSet": "130615726366949784",
  "scriptPath": null,
  "badPasswordTime": "0",
  "employeeID": null,
  "badPwdCount": "0",
  "accountExpirationDate": null,
  "userPrincipalName": "steve.carter@ad0.example.com",
  "sAMAccountName": "steve.carter",
  "mail": "steven.carter@example.com",
  "logonCount": "0",
}
```

```
{
  "cannotChangePassword": false,
  "division": null,
  "streetAddress": null,
  "allowReversiblePasswordEncryption": false,
  "description": null,
  "whenChanged": "11/27/2014 3:37:16 PM",
  "title": null,
  "lastLogon": "0",
  "company": null,
  "homeDirectory": null,
  "whenCreated": "6/23/2014 2:50:48 PM",
  "givenName": "Steven",
  "telephoneNumber": "555-2518",
  "homeDrive": null,
  "uSNCreated": "20912",
  "smartcardLogonRequired": false,
  "distinguishedName": "CN=Steven Carter,CN=Users,DC=example,DC=com",
  "createTimeStamp": "6/23/2014 2:50:48 PM",
  "department": null,
  "memberOf": [
    "CN=employees,DC=example,DC=com"
  ],
  "homePhone": null
}
```

4. Test that you can authenticate as one of the users in your Active Directory. The username that you specify here can be either an ObjectGUID, UPN, sAMAccountname or CN:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0" `
--header "Content-Type: application/json" `
--request POST `
--data "{
  \"username\" : \"Steven Carter\",
  \"password\" : \"Passw0rd\"
}" `
"http://localhost:8080/openidm/system/adpowershell/account?_action=authenticate"
{
  "_id": "6feef4a0-b121-43dc-be68-a96703a49aba"
}
```

The request returns the ObjectGUID if the authentication is successful.

5. You can return the complete record for a specific user, using the query filter syntax described in "Construct Queries" in the *Object Modeling Guide*.

The following query returns the record for the guest user:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0" `
--request GET `
"http://localhost:8080/openidm/system/adpowershell/account?_queryFilter=cn=eq+'guest'"
{
```

```
"remainingPagedResults": -1,
"pagedResultsCookie": null,
"resultCount": 1,
"result": [
  {
    "_id": "f2e08a5c-473f-4798-a2d5-d5cc27c862a9",
    "postalCode": null,
    "passwordNotRequired": true,
    "cn": "Guest",
    "name": "Guest",
    "trustedForDelegation": false,
    "uSNChanged": "8197",
    "manager": null,
    "objectGUID": "f2e08a5c-473f-4798-a2d5-d5cc27c862a9",
    "modifyTimeStamp": "6/9/2014 12:35:16 PM",
    "employeeNumber": null,
    "userAccountControl": 66082,
    "whenChanged": "6/9/2014 12:35:16 PM",
    "initials": null,
    "pwdLastSet": "0",
    "scriptPath": null,
    "badPasswordTime": "0",
    "employeeID": null,
    "badPwdCount": "0",
    "accountExpirationDate": null,
    "sAMAccountName": "Guest",
    "logonCount": "0",
    "cannotChangePassword": true,
    "division": null,
    "streetAddress": null,
    "allowReversiblePasswordEncryption": false,
    "description": "Built-in account for guest access to the computer/domain",
    "userPrincipalName": null,
    "title": null,
    "lastLogon": "0",
    "company": null,
    "homeDirectory": null,
    "whenCreated": "6/9/2014 12:35:16 PM",
    "givenName": null,
    "homeDrive": null,
    "uSNCreated": "8197",
    "smartcardLogonRequired": false,
    "distinguishedName": "CN=Guest,CN=Users,DC=example,DC=com",
    "createTimeStamp": "6/9/2014 12:35:16 PM",
    "department": null,
    "memberOf": [
      "CN=Guests,CN=Builtin,DC=example,DC=com"
    ],
    "homePhone": null,
    "displayName": null,
    "passwordNeverExpires": true
  }
]
}
```

- Test that you can create a user on the Active Directory server by sending a POST request with the **create** action.

The following request creates the user **Jane Doe** on the Active Directory server:

```
curl.exe `
--header "X-OpenIDM-Username: openid-admin" `
--header "X-OpenIDM-Password: openid-admin" `
--header "Accept-API-Version: resource=1.0" `
--header "Content-Type: application/json" `
--request POST `
--data '{
  \"distinguishedName\" : \"CN=Jane Doe,CN=Users,DC=EXAMPLE,DC=COM\",
  \"sn\" : \"Doe\",
  \"cn\" : \"Jane Doe\",
  \"sAMAccountName\" : \"sample\",
  \"userPrincipalName\" : \"janedoe@example.com\",
  \"enabled\" : true,
  \"password\" : \"Passw0rd\",
  \"telephoneNumber\" : \"0052-611-091\"
}' `
"http://localhost:8080/openidm/system/adpowershell/account?_action=create"
{
  "id": "42725210-8dce-4fdf-b0e0-393cf0377fdf",
  "title": null,
  "uSNCreated": "47244",
  "pwdLastSet": "130615892934093041",
  "cannotChangePassword": false,
  "telephoneNumber": "0052-611-091",
  "smartcardLogonRequired": false,
  "badPwdCount": "0",
  "department": null,
  "distinguishedName": "CN=Jane Doe,CN=Users,DC=example,DC=com",
  "badPasswordTime": "0",
  "employeeID": null,
  "cn": "Jane Doe",
  "division": null,
  "description": null,
  "userPrincipalName": "janedoe@example.com",
  "passwordNeverExpires": false,
  "company": null,
  "memberOf": [],
  "givenName": null,
  "streetAddress": null,
  "sn": "Doe",
  "initials": null,
  "logonCount": "0",
  "homeDirectory": null,
  "employeeNumber": null,
  "objectGUID": "42725210-8dce-4fdf-b0e0-393cf0377fdf",
  "manager": null,
  "lastLogon": "0",
  "trustedForDelegation": false,
  "scriptPath": null,
  "allowReversiblePasswordEncryption": false,
  "modifyTimeStamp": "11/27/2014 8:14:53 PM",
  "whenCreated": "11/27/2014 8:14:52 PM",
  "whenChanged": "11/27/2014 8:14:53 PM",
  "accountExpirationDate": null,
  "name": "Jane Doe",
  "displayName": null,
```

```
{
  "homeDrive": null,
  "passwordNotRequired": false,
  "createTimeStamp": "11/27/2014 8:14:52 PM",
  "uSNChanged": "47248",
  "sAMAccountName": "sample",
  "userAccountControl": 512,
  "homePhone": null,
  "postalCode": null
}
```

7. Test that you can update a user object on the Active Directory server by sending a PUT request with the complete object, including the user ID in the URL.

The following request updates user **Jane Doe**'s entry, including her ID in the request. The update sends the same information that was sent in the **create** request, but adds an **employeeNumber**:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0" `
--header "Content-Type: application/json" `
--header "If-Match: *" `
--request PUT `
--data '{
  \"distinguishedName\" : \"CN=Jane Doe,CN=Users,DC=EXAMPLE,DC=COM\",
  \"sn\" : \"Doe\",
  \"cn\" : \"Jane Doe\",
  \"userPrincipalName\" : \"janedoe@example.com\",
  \"enabled\" : true,
  \"password\" : \"Passw0rd\",
  \"telephoneNumber\" : \"0052-611-091\",
  \"employeeNumber\" : \"567893\"
}' `
"http://localhost:8080/openidm/system/adpowershell/account/42725210-8dce-4fdf-b0e0-393cf0377fdf"
{
  "_id": "42725210-8dce-4fdf-b0e0-393cf0377fdf",
  "title": null,
  "uSNCreated": "47244",
  "pwdLastSet": "130615906375709689",
  "cannotChangePassword": false,
  "telephoneNumber": "0052-611-091",
  "smartcardLogonRequired": false,
  "badPwdCount": "0",
  "department": null,
  "distinguishedName": "CN=Jane Doe,CN=Users,DC=example,DC=com",
  "badPasswordTime": "0",
  "employeeID": null,
  "cn": "Jane Doe",
  "division": null,
  "description": null,
  "userPrincipalName": "janedoe@example.com",
  "passwordNeverExpires": false,
  "company": null,
  "memberOf": [],
  "givenName": null,
  "streetAddress": null,
  "sn": "Doe",
  "initials": null,
```

```
"logonCount": "0",
"homeDirectory": null,
"employeeNumber": "567893",
"objectGUID": "42725210-8dce-4fdf-b0e0-393cf0377fdf",
"manager": null,
"lastLogon": "0",
"trustedForDelegation": false,
"scriptPath": null,
"allowReversiblePasswordEncryption": false,
"modifyTimeStamp": "11/27/2014 8:37:17 PM",
"whenCreated": "11/27/2014 8:14:52 PM",
"whenChanged": "11/27/2014 8:37:17 PM",
"accountExpirationDate": null,
"name": "Jane Doe",
"displayName": null,
"homeDrive": null,
"passwordNotRequired": false,
"createTimeStamp": "11/27/2014 8:14:52 PM",
"uSNChanged": "47253",
"sAMAccountName": "sample",
"userAccountControl": 512,
"homePhone": null,
"postalCode": null
}
```

8. Test whether you are able to delete a user object on the Active Directory server by sending a DELETE request with the user ID in the URL.

The following request deletes user **Jane Doe**'s entry:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0" `
--request DELETE `
"http://localhost:8080/openidm/system/adpowershell/account/42725210-8dce-4fdf-b0e0-393cf0377fdf"
```

The response includes the complete user object that was deleted.

You can attempt to query the user object to confirm that it has been deleted:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0" `
--request GET `
"http://localhost:8080/openidm/system/adpowershell/account/42725210-8dce-4fdf-b0e0-393cf0377fdf"
{
  "message": "",
  "reason": "Not Found",
  "code": 404
}
```

Chapter 21

Connect to a MySQL Database With ScriptedSQL

This sample uses the Groovy Connector Toolkit bundled with IDM ([openidm/connectors/groovy-connector-1.5.20.8.jar](#)) to implement a ScriptedSQL connector that interacts with an external MySQL database (HRDB), and also demonstrates the following functionality:

- Complex data types.

Complex data types can be stored, retrieved and synchronized like any other object property. They are stored in the managed data as JSON objects, represented as a string, but can be mapped to external resources in any format required. You can customize the mapping to do additional work with or transformations on the complex data types.

This sample defines one complex data type, `cars`, discussed in more detail later in this section.

- Event hooks to perform actions.

The mapping from the internal repository to the external `hrdb` database includes two script hooks. The first hook is for an `onCreate` event and the second is for an `onUpdate` event.

- Custom scripted endpoints in the *Scripting Guide*.

Custom scripted endpoints are configured in the provisioner configuration file and allow you to execute custom scripts over REST. This sample uses a custom scripted endpoint to reset the database and populate it with data.

Caution

Because MySQL cannot "un-hash" user passwords there is no way for a reconciliation operation to retrieve the password from MySQL and store it in the managed user object. This issue might impact configurations that support multiple external resources in that passwords might not be synchronized immediately after reconciliation from MySQL to the managed user repository. Users who are missing in the repository will be created by reconciliation but their passwords will be empty. When those users are synchronized to other external resources, they will have empty passwords in those resources. Additional scripting might be required to handle this situation, depending on the requirements of your deployment.

The Groovy scripts required for the sample are located in the `samples/scripted-sql-with-mysql/tools` directory. Note that the power of the Groovy connector is in the associated Groovy scripts, and their application in your particular deployment. The scripts provided with this sample are specific to the

sample. You must customize these scripts to address the requirements of your specific deployment. The sample scripts are a good starting point on which to base your customization.

Configure the External MySQL Database

This sample assumes a database running on the localhost.

1. Download MySQL Connector/J version 5.1 or later.
2. Unpack the downloaded file, and copy the `.jar` file to `openidm/lib`:

```
cp mysql-connector-java-version-bin.jar /path/to/openidm/lib/
```

Note

If you are running this sample with an SQL database other than MySQL, download the corresponding driver and place it in the `openidm/lib` directory. It is not necessary to create an OSGi bundle for the driver.

3. Set up MySQL to listen on localhost, port `3306`. IDM will connect to the `hrdb` database as user `root` with password `password`.

To use an existing MySQL instance that runs on a different host or port, or to change the database credentials, edit the `configurationProperties` in the connector configuration file (`samples/scripted-sql-with-mysql/conf/provisioner.openicf-hrdb.json`) before you start the sample. The default configuration is as follows:

```
"configurationProperties" : {  
  "username" : "root",  
  "password" : "password",  
  "driverClassName" : "com.mysql.jdbc.Driver",  
  "url" : "jdbc:mysql://localhost:3306/hrdb?serverTimezone=UTC",  
  ...  
}
```

Note

If you are running a relatively recent version of MySQL (5.5.45+, 5.6.26+, 5.7.6+), the default configuration expects SSL, *which is strongly advised in a production environment*. If you are running this in a test environment, you can bypass the SSL requirement:

- Add `&useSSL=false` to the end of the `url`.
- If you are running MySQL 8.0.11+, add `&allowPublicKeyRetrieval=true` to the end of the `url`.

4. Set up the `hrdb` database, with which IDM will synchronize its managed user repository:

```
mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.13 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> CREATE DATABASE hrdb CHARACTER SET utf8 COLLATE utf8_bin;
Query OK, 1 row affected (0.00 sec)
```

5. Configure your GRANT permissions:

- For MySQL 5.6 and lower, run the following statement:

```
GRANT ALL ON hrdb.* TO root@'%' IDENTIFIED BY 'password';
```

- For MySQL 5.7 and higher, run the following statements:

```
CREATE USER IF NOT EXISTS 'root'@'%' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON hrdb.* TO 'root'@'%' WITH GRANT OPTION;
```

Run the Sample

The mapping configuration file ([sync.json](#)) for this sample includes the mapping `systemHrdb_managedUser`. You will use this mapping to synchronize users from the source `hrdb` database with the target IDM repository.

- Start IDM with the configuration for the ScriptedSQL sample:

```
/path/to/openidm/startup.sh -p samples/scripted-sql-with-mysql
```

- Run the custom script described in the previous section to reset the database and populate it with sample data.

You can run the script again, at any point, to reset the database.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/hrdb?_action=script&scriptId=ResetDatabase"
{
  "actions": [
    {
      "result": "Database reset successful."
    }
  ]
}
```

The **hrdb** database should now be populated with sample data.

3. Review the contents of the database:

```
mysql -u root -p
Enter password:
...
mysql > use hrdb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql > select * from users;
```

id	uid	password	firstname	lastname	fullname	email
1	bob	e38ad2149...	Bob	Fleming	Bob Fleming	Bob.Fle...
2	rowley	2aa60a8ff...	Rowley	Birkin	Rowley Birkin	Rowley...
3	louis	1119cfd37...	Louis	Balfour	Louis Balfour	Louis.B...
4	john	ald7584da...	John	Smith	John Smith	John.Sm...
5	jdoe	edba955d0...	John	Doe	John Doe	John.Do...

5 rows in set (0.00 sec)

Note

The passwords in the output shown above are hashed to the SHA-1 standard because as they cannot be read into IDM as clear text. The SHA-1 Hash function is used for compatibility reasons. Use a more secure algorithm in a production database.

4. Reconcile the hrdb database with the managed user repository.

- To reconcile the repository by using the Administration UI:
 - Log in to the Admin UI at the URL <https://localhost:8443/admin> as the default administrative user (**openidm-admin**) with password **openidm-admin**.

2. Select Configure > Mappings.

The Mappings page shows two mappings, one from the **hrdb** database to the IDM managed user repository (**managed/user**), and one in the opposite direction.

3. Click Reconcile on the first mapping (systemHrdb_managedUser).

- To reconcile the repository by using the command-line, launch the reconciliation operation with the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
action=recon&mapping=systemHrdb_managedUser&waitForCompletion=true"
{
  "state": "SUCCESS",
  "_id": "f3c618aa-cc3b-49ed-9a3a-00b012db2513"
}
```

The reconciliation operation creates the five users from the MySQL database in the IDM repository.

5. Retrieve the list of users from the repository.

- To retrieve the users in the repository from the Admin UI:

1. Select Manage > User to display the User List.

The five users from the **hrdb** database have been reconciled to the OpenIDM repository.

2. To retrieve the details of a specific user, click that user entry.

- To retrieve the users from the repository by using the command-line, query the IDs in the repository as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryId=query-all-ids"
{
  "result": [
    {
      "_id": "1b379e4d-3b8d-47e7-93d5-a72c4b483e39",
      "_rev": "000000002d93e471"
    },
    {
      "_id": "a658f751-d6e9-4b5d-af56-071a9b05c3af",
      "_rev": "000000003c83d48a"
    }
  ],
}
```



```
{
  "_id": "5b31027b-09f8-4c7f-abfa-c6bc86ae3943",
  "_rev": "00000000b042e559"
},
{
  "_id": "1b3f6b06-1752-4c40-ba34-51d30b184b9d",
  "_rev": "0000000092bdda6d"
},
{
  "_id": "9c62f0d2-47e2-4fc5-89d1-b50b782b1022",
  "_rev": "0000000025cdd3c6"
}
],
"resultCount": 5,
"pagedResultsCookie": null,
"totalPagedResultsPolicy": "NONE",
"totalPagedResults": -1,
"remainingPagedResults": -1
}
```

To retrieve a complete user record, query the `userName` of the individual user entry. The following query returns the record for the user **Rowley Birkin**:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/?_queryId=for-userName&uid=rowley"

"result": [
  {
    "_id": "1b379e4d-3b8d-47e7-93d5-a72c4b483e39",
    "_rev": "000000002d93e471",
    "mail": "Rowley.Birkin@example.com",
    "userName": "rowley",
    "sn": "Birkin",
    "organization": "SALES",
    "givenName": "Rowley",
    "cars": [
      {
        "year": "2013",
        "make": "BMW",
        "model": "328ci"
      },
      {
        "year": "2010",
        "make": "Lexus",
        "model": "ES300"
      }
    ]
  },
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
],
...
}
```

Regardless of how you have retrieved Rowley Birkin's entry, note the `cars` property in this user's entry. This property demonstrates a complex object, stored in JSON format in the user entry, as a list that contains multiple objects. In the MySQL database, the `car` table joins to the `users` table through a `cars.users_id` column. The Groovy scripts read this data from MySQL and repackage it in a way that IDM can understand. With support for complex objects, the data is passed through to IDM as a list of `car` objects. Data is synchronized from IDM to MySQL in the same way. Complex objects can also be nested to any depth.

Group membership (not demonstrated here) is maintained with a traditional "join table" in MySQL (`groups_users`). IDM does not maintain group membership in this way, so the Groovy scripts do the work to translate membership between the two resources.

Test the Event Hooks

This sample uses the `onCreate` and `onUpdate` hooks to log messages when a user is created or updated in the external database.

The sample's `conf/sync.json` file defines these event hooks as follows:

```
...
{
  "name" : "managedUser_systemHrdb",
  "source" : "managed/user",
  "target" : "system/hrdb/account",
  "links" : "systemHrdb_managedUser",
  "correlationQuery" : {
    "type" : "text/javascript",
    "source" : "({'_queryFilter': 'uid eq \'' + source.userName + '\''});"
  },
  "onCreate" : {
    "type" : "text/javascript",
    "source" : "logger.info(\"Creating new user in external repo\")"
  },
  "onUpdate" : {
    "type" : "text/javascript",
    "source" : "logger.info(\"Updating existing user in external repo\")"
  },
  ...
}
```

Using these event hooks, IDM logs a message when a user is created or updated in the external database. In this sample, the script source is included in the mapping. However, a script can also be called from an external file. For more information about event hooks, see *"Script Triggers"* in the *Scripting Guide*.

To test the event hooks, create a new managed user as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "mail": "fdoe@example.com",
  "sn": "Doe",
  "telephoneNumber": "555-1234",
  "userName": "fdoe",
  "givenName": "Felicitas",
  "description": "Felicitas Doe",
  "displayName": "fdoe"}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "_id": "2e5e9748-77e6-4019-90e1-abe9ab897343",
  "_rev": "0000000015b2d4ba",
  "mail": "fdoe@example.com",
  "sn": "Doe",
  "telephoneNumber": "555-1234",
  "userName": "fdoe",
  "givenName": "Felicitas",
  "description": "Felicitas Doe",
  "displayName": "fdoe",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

The implicit synchronization between the managed user repository and the HRDB database creates that user in the database automatically.

Check the latest log file at `path/to/openidm/logs/openidm0.log.0`. You should see the following message at the end of the log:

```
INFO: Creating new user in external repo
```

Query the new user entry in the HRDB database:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/hrdb/account?_queryFilter=uid+eq+'fdoe'"
{
  "result": [
    {
      "_id": "6",
      "cars": [],
      "firstName": "Felicitas",
      "uid": "fdoe",
      "lastName": "Doe",
      "organization": "IDM",
      "fullName": "Felicitas Doe",
      "email": "fdoe@example.com"
    }
  ],
  ...
}
```

Update fdoe's entry in the HRDB database with a patch request. The following request updates the user's **organization** field:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
  "operation" : "replace",
  "field" : "organization",
  "value" : "example.com"
} ]' \
"http://localhost:8080/openidm/system/hrdb/account/6"
{
  "_id": "6",
  "cars": [],
  "firstName": "Felicitas",
  "uid": "fdoe",
  "lastName": "Doe",
  "organization": "example.com",
  "fullName": "Felicitas Doe",
  "email": "fdoe@example.com"
}
```

Note that this update does not reference the **onUpdate** script hook so this change is not logged in `openidm0.log.0`.

Run the Sample With Paging

All ICF connectors from version 1.4 onwards support the use of paging parameters to restrict query results. The following command indicates that only two records should be returned (`_pageSize=2`) and that the records should be sorted according to their `timestamp` and `_id` (`_sortKeys=timestamp,_id`). Including the `timestamp` in the sort ensures that, as you page through the set, changes to records that have already been visited are not lost. Instead, those records are pushed onto the last page:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/hrdb/account?_queryFilter=true&_pageSize=2&_sortKeys=timestamp,_id"
{
  "result": [
    {
      "_id": "1",
      "firstName": "Bob",
      "cars": [
        {
          "year": "1979",
          "make": "Ford",
          "model": "Pinto"
        }
      ],
      "fullName": "Bob Fleming",
      "email": "Bob.Fleming@example.com",
      "uid": "bob",
      "lastName": "Fleming",
      "organization": "HR"
    },
    {
      "_id": "2",
      "firstName": "Rowley",
      "cars": [
        {
          "year": "2013",
          "make": "BMW",
          "model": "328ci"
        }
      ],
      "fullName": "Rowley Birkin",
      "email": "Rowley.Birkin@example.com",
      "uid": "rowley",
      "lastName": "Birkin",
      "organization": "SALES"
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": "2018-04-05+16%3A30%3A22.0%2C2",
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

The `pagedResultsCookie` is used by the server to keep track of the position in the search results. You can ignore the `"remainingPagedResults": -1` in the output. The real value of this property is not returned because the scripts that the connector uses do not do any counting of the records in the resource.

Using the `pagedResultsCookie` from the previous step, run a similar query, to retrieve the following set of records in the database:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/hrdb/account?_queryId=query-all-ids&_pageSize=2&_sortKeys=timestamp,_id&_pagedResultsCookie=2018-04-05+16%3A30%3A22.0%2C2"
{
  "result": [
    {
      "_id": "3",
    },
    {
      "_id": "4",
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": "2018-04-05+16%3A30%3A22.0%2C4",
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

For more information about paging, see "Page Query Results" in the *Object Modeling Guide*.

Chapter 22

Direct Audit Information To MySQL

The sample includes an external CSV file and a mapping between objects in that file and the managed user repository. The reconciliations across this mapping generate the audit records that will be directed to the MySQL database. The connection to the MySQL database is through a ScriptedSQL implementation of the Groovy Connector Toolkit.

About the Configuration Files

The files that demonstrate the functionality of this sample are located under `/path/to/openidm/samples/audit-jdbc/`, in the `conf/` and `data/` directories.

The following files play important roles in this sample:

`conf/provisioner.openicf-auditdb.json`

This file provides the configuration for the Scripted SQL implementation of the Groovy Connector. The file specifies, among other things, the connection details to the MySQL database, the connector version information, and the object types that are supported for this connection. For more information, see "Groovy Connector Toolkit" in the *Connectors Guide*.

`conf/provisioner.openicf-csvfile.json`

This file provides the configuration for this instance of the CSV connector. It includes, among other things, the location of the CSV file resource.

`conf/sync.json`

Provides the mapping between managed users and the data set in the CSV file.

`conf/audit.json`

This file configures the router as the audit event handler, and routes audit logs to a remote system, identified as `auditdb`.

`data/csvConnectorData.csv`

This file contains the sample data set that will be reconciled to the managed user repository.

`data/sample_audit_db.mysql`

This file sets up the schema for the MySQL database that will contain the audit logs.

`tools/*.groovy`

The Groovy scripts in this directory allow the connector to perform operations on the MySQL database.

Configure the MySQL Database

Before you start this sample, MySQL must be installed and running, and must include the database required for the sample. In addition, IDM must include the connector .jar required to connect to the MySQL database.

The sample assumes the following MySQL configuration:

- You can connect to the MySQL database over the network, as user `root` with password `password`.
1. Install and configure MySQL.
 2. When MySQL is up and running, import the database schema to set up the database required for the sample:

```
mysql -u root -p < /path/to/openidm/samples/audit-jdbc/data/sample_audit_db.mysql
Enter password: password
```

This step sets up an `audit` database with six tables that correspond to the various audit events. You can view the tables in the audit database as follows:

```
mysql -u root -p
Enter password: password
mysql> use audit
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_audit |
+-----+
| auditaccess      |
| auditactivity    |
| auditauthentication |
| auditconfig      |
| auditrecon       |
| auditsync        |
+-----+
6 rows in set (0.00 sec)
```

3. Download [MySQL Connector/J](#), version 5.1 or later from the MySQL website. Unpack the delivery, and copy the .jar into the `openidm/bundle` directory:

```
cp mysql-connector-java-version-bin.jar /path/to/openidm/bundle/
```

4. Edit the `url` property in the SQL connector configuration file (`conf/provisioner.openicf-auditdb.json`) to match the host and port of your MySQL instance. For example:


```
"url" : "jdbc:mysql://localhost:3306/audit?serverTimezone=UTC",
```

Note

If you are running a relatively recent version of MySQL (5.5.45+, 5.6.26+, 5.7.6+), the default configuration expects SSL, *which is strongly advised in a production environment*. If you are running this in a test environment, you can bypass the SSL requirement:

- Add `&useSSL=false` to the end of the `url`.
- If you are running MySQL 8.0.11+, add `&allowPublicKeyRetrieval=true` to the end of the `url`.

Run the Sample

In this section, you will start IDM, then run a reconciliation between the CSV file and the managed user repository. After the reconciliation, you should be able to read the audit logs in the `audit` database on your MySQL instance.

1. Prepare IDM as described in "Prepare IDM", then start the server with the configuration for this sample:

```
/path/to/openidm/startup.sh -p samples/audit-jdbc
```

2. Reconcile the two data sources, either over the command-line or by using the Admin UI.

To run the reconciliation over REST, use the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemCsvfileAccounts_managedUser&waitForCompletion=true"
{
  "_id": "a3664c26-bf82-4100-b411-19edc248c306-7",
  "state": "SUCCESS"
}
```

To run the reconciliation from the Admin UI, select Configure > Mappings, select the `systemCsvfileAccounts_managedUser` mapping, then select Reconcile.

3. Inspect the tables in the `audit` database to see how the logs have been routed to that location.

The following example displays the reconciliation audit logs:

```
mysql -u root -p
Enter password: password
...
mysql> use audit;
...
mysql> show tables;
+-----+
| Tables_in_audit |
+-----+
| auditaccess      |
| auditactivity    |
| auditauthentication |
| auditconfig      |
| auditrecon       |
| auditsync        |
+-----+
6 rows in set (0.01 sec)
mysql> select * from auditrecon;

+----+-----+-----+-----+-----+-----+-----+-----+
| id | objectid | transactionid | activitydate | eventname | userid      | ... | mapping... |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | a3664... | a3664c26-b... | 2017-02-1... | recon     | openidm-admin | ... | systemCsvfile... |
+----+-----+-----+-----+-----+-----+-----+-----+

```

You can inspect the other audit logs in the same way.

4. By default, the audit configuration in this sample uses the router audit handler for queries, as indicated in the following line from the `conf/audit.json` file:

```
"handlerForQueries" : "router",
```

With this configuration, when you query the audit logs over REST, the audit data is returned from the router handler (in this case the MySQL database). The following example shows how to query the reconciliation audit log:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/audit/recon?_queryFilter=true"
{
  "result": [
    {
      "_id": "a3664c26-bf82-4100-b411-19edc248c306-16",
      "action": null,
      "trackingIds": null,
      "sourceObjectId": null,
      "timestamp": "2017-02-14T10:19:30.688Z",
      "ambiguousTargetObjectIds": null,
      "reconId": "a3664c26-bf82-4100-b411-19edc248c306-7",
      "status": null,
      "eventName": "recon",
      "mapping": "systemCsvfileAccounts_managedUser",
      "targetObjectId": null,
      "situation": null,
      "userId": "openidm-admin",
      "transactionId": "a3664c26-bf82-4100-b411-19edc248c306-7",
      "message": "Reconciliation initiated by openidm-admin",
    }
  ]
}
```

```
    "reconAction": "recon",  
    "messageDetail": null,  
    "entryType": "start",  
    "linkQualifier": null,  
    "reconciling": null,  
    "exception": null  
  },  
  ...  
}
```

You can query the other [audit logs](#) in the same way.

Chapter 23

Direct Audit Information to a JMS Broker

This sample shows how to configure a Java Message Service (JMS) audit event handler to direct audit information to a JMS broker.

JMS is an API that supports Java-based peer-to-peer messages between clients. The JMS API can create, send, receive, and read messages, reliably and asynchronously. You can set up a JMS audit event handler to publish messages that comply with the Java Message Service Specification Final Release 1.1.

This sample demonstrates the use of the JMS audit event handler. In the sample you will set up communication between IDM and an external JMS Message Broker, as well as Apache Active MQ as the JMS provider and message broker.

Note

JMS topics are not related to ForgeRock audit event topics. The ForgeRock implementation of JMS topics uses the publish/subscribe messaging domain to direct messages to the JMS audit event handler. In contrast, ForgeRock audit event topics specify categories of events.

Dependencies for JMS Messaging

The JMS audit event handler requires ActiveMQ, and a number of dependencies. This section lists the dependencies, where you can download them, and where they must be installed in the IDM instance. If you use a different ActiveMQ version, the dependency versions may differ from those shown.

Download the following files:

- ActiveMQ binary. This sample was tested with ActiveMQ 5.15.13.
- ActiveMQ Client that corresponds to your ActiveMQ version.
- JmDNS, version 3.4.1.
- The most recent `bnd` JAR file from <https://repo1.maven.org/maven2/biz/aQute/bnd/biz.aQute.bnd/>. The `bnd` utility lets you create OSGi bundles for libraries that do not yet support OSGi.
- Apache Geronimo J2EE management bundle.
- `hawtbuf-1.11` JAR file (a Maven-based protocol buffer compiler).

1. Unpack the ActiveMQ binary. For example:

```
tar -zxvf ~/Downloads/apache-activemq-5.15.13-bin.tar.gz
```

2. Create a temporary directory, move the ActiveMQ Client, and `bnd` JAR files to that directory, then change to that directory:

```
mkdir ~/Downloads/tmp
mv activemq-client-5.15.13.jar ~/Downloads/tmp/
mv biz.aQute.bnd-version.jar ~/Downloads/tmp/
cd ~/Downloads/tmp/
```

3. Create an OSGi bundle as follows:

- a. In a text editor, create a BND file named `activemq.bnd` and save it to the current directory. The file should have the following contents:

```
version=5.15.13
Export-Package: *;version=${version}
Bundle-Name: ActiveMQ :: Client
Bundle-SymbolicName: org.apache.activemq
Bundle-Version: ${version}
```

Your `tmp/` directory should now contain the following files:

```
ls ~/Downloads/tmp/
activemq-client-5.15.13.jar  activemq.bnd  biz.aQute.bnd-version.jar
```

- b. In that same directory, create the OSGi bundle archive file as follows:

```
java -jar biz.aQute.bnd-version.jar \
wrap --properties activemq.bnd \
--output activemq-client-5.15.13-osgi.jar \
activemq-client-5.15.13.jar
```

4. Copy the resulting `activemq-client-5.15.13-osgi.jar` file to the `openidm/bundle` directory:

```
cp activemq-client-5.15.13-osgi.jar /path/to/openidm/bundle/
```

5. Copy the *Apache Geronimo*, *hawtbuf*, and *JmDNS* JAR files to the `openidm/bundle` directory:

```
cp ~/Downloads/geronimo-j2ee-management_1.1_spec-1.0.1.jar /path/to/openidm/bundle/
cp ~/Downloads/hawtbuf-1.11.jar /path/to/openidm/bundle/
cp ~/Downloads/jmdns-3.4.1.jar /path/to/openidm/bundle
```

Your IDM instance is now ready for you to configure the JMS audit event handler.

Configure SSL for ActiveMQ

This configuration provides a connection to the ActiveMQ server instance with TLSv1.3.

1. In the directory where you unpacked ActiveMQ, edit the `conf/activemq.xml` file as follows:

- In the `<brokers>` element, add an `<sslContext>`:

```
<broker xmlns="http://activemq.apache.org/schema/
core" brokerName="localhost" dataDirectory="${activemq.data}">
  ...
  <sslContext>
    <sslContext keyStore="file:${activemq.conf}/broker.ks" keyStorePassword="password"/>
  </sslContext>
  ...
</broker>
```

- In the `<transportConnectors>` element, add an `ssl <transportConnector>`:

```
<transportConnectors>
  ...
  <transportConnector name="ssl" uri="ssl://0.0.0.0:61617?transport.needClientAuth=false"/>
</transportConnectors>
```

Note

To enable mutual authentication, set `transport.needClientAuth=true`, and import the IDM server certificate into the ActiveMQ truststore (`conf/broker.ts`).

2. Delete the existing self-signed server certificate from the ActiveMQ keystore and truststore:

```
keytool \
-delete \
-keystore /path/to/activeMQ/conf/broker.ts \
-alias broker-localhost
Enter keystore password: password
keytool \
-delete \
-keystore /path/to/activeMQ/conf/broker.ks \
-alias broker-localhost
Enter keystore password: password
```

3. Generate a new self-signed server certificate for ActiveMQ:

```
keytool \
-genkey \
-keyalg RSA \
-alias broker-localhost \
-keystore /path/to/activeMQ/conf/broker.ks \
-storepass password \
-validity 360 \
-keysize 2048
```

Important

The **CN** in the generated self-signed certificate *must* match the hostname that you specify in the IDM JMS provider URL. If you are using `localhost` to connect to the broker, you must specify `localhost` when **keytool**

prompts you for the **first and last name**. If the **CN** is not the same as the hostname, the server certificate validation will fail.

- Export the ActiveMQ server certificate:

```
keytool \  
-export \  
-alias broker-localhost \  
-file broker-localhost.key \  
-keystore /path/to/activeMQ/conf/broker.ks  
Enter keystore password: password  
Certificate stored in file <broker-localhost.key>
```

- Import the ActiveMQ server certificate into the IDM truststore:

```
keytool \  
-import \  
-alias activemq \  
-keystore /path/to/openidm/security/truststore \  
-file broker-localhost.key  
Enter keystore password: changeit  
Owner: CN=localhost, OU=Unknown, O=example.com, L=Unknown, ST=Unknown, C=Unknown  
Issuer: CN=localhost, OU=Unknown, O=example.com, L=Unknown, ST=Unknown, C=Unknown  
...  
Trust this certificate? [no]: yes  
Certificate was added to keystore
```

Configure a Secure Port for JMS Messages

Edit `/path/to/openidm/samples/audit-jms/conf/audit.json`, as follows:

```
"java.naming.provider.url" : "ssl://localhost:61617?daemon=true"
```

Start the ActiveMQ Broker and IDM

With the appropriate bundles in the `/path/to/openidm/bundle/` directory, you can start the ActiveMQ message broker, then start IDM with the configuration for this sample.

- Navigate to the directory where you unpacked the ActiveMQ binary and run the following command to start the ActiveMQ broker:

```
cd ~/Downloads/apache-activemq-5.15.13  
bin/activemq start  
INFO: Loading '/path/to/Downloads/apache-activemq-5.15.13/bin/env'  
INFO: Using java '/usr/bin/java'  
INFO: Starting - inspect logfiles specified in logging.properties and log4j.properties to get details  
INFO: pidfile created : '/path/to/Downloads/apache-activemq-5.15.13/data/activemq.pid' (pid '69627')
```

- Start IDM with the sample configuration:

```
cd /path/to/openidm/
./startup.sh -p samples/audit-jms
```

- Go to the ActiveMQ Console at <http://localhost:8161/admin/topics.jsp>, and verify audit messages are being created and sent to the `forgerock.idm.audit` audit topic.

Note

If you see the following error in the OSGi console, make sure that you have installed all the required dependencies and that you have started the ActiveMQ broker.

```
SEVERE: Unable to create JmsAuditEventHandler 'jms': null
```

Configure and Use a JMS Consumer Application

To take advantage of the Apache ActiveMQ event broker, the JMS audit sample includes a Java consumer in the following directory: `/path/to/openidm/samples/audit-jms/consumer/`

Assuming you have Apache Maven installed on the local system, you can compile that sample consumer with the following commands:

```
cd /path/to/openidm/samples/audit-jms/consumer/
mvn clean install
```

When the build process is complete, you'll see a **BUILD SUCCESS** message:

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 22.852 s
[INFO] Finished at: 2017-02-17T17:21:35+02:00
[INFO] Final Memory: 18M/148M
[INFO] -----
```

Note

You might see **[WARNING]** messages during the build. As long as the messages end with **BUILD SUCCESS**, you can proceed with the JMS consumer application.

When the consumer is compiled, run the following command in the same directory to output audit messages related to IDM actions:


```
mvn \
exec:java \
-Dexec.mainClass="consumer.src.main.java.SimpleConsumer" \
-Dexec.args="tcp://localhost:61616"
[INFO]
[INFO] -----
[INFO] Building SimpleConsumer 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- exec-maven-plugin:1.5.0:java (default-cli) @ SimpleConsumer ---
Downloading: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/...
Downloaded: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/...
Downloading: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/...
Downloaded: https://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/...
Connection factory=org.apache.activemq.ActiveMQConnectionFactory
READY, listening for messages. (Press 'Enter' to exit)
```

Look for the message **READY, listening for messages**.

If you've configured ActiveMQ on a secure port, run this command:

```
MAVEN_OPTS="-Djavax.net.ssl.trustStore=/path/to/openidm/security/truststore" \
mvn \
exec:java \
-Dexec.mainClass="consumer.src.main.java.SimpleConsumer" \
-Dexec.args="ssl://localhost:61617?daemon=true"
```

Try some actions on IDM, either in a different console or in the Admin UI. Watch the output in the **SimpleConsumer** console. For example, you might see output similar to the following when you run a reconciliation on the data in this sample:

```
{
  "auditTopic": "recon",
  "event" :{
    "transactionId": "f3e108a9-eb75-4ec8-96c5-9dc2f5137d51-176",
    "timestamp": "2017-02-17T15:25:34.899Z",
    "eventName": "recon",
    "userId": "openidm-admin",
    "exception": null,
    "linkQualifier": null,
    "mapping": "systemCsvfileAccounts_managedUser",
    "message": "SOURCE_IGNORED: 0 UNASSIGNED: 0 AMBIGUOUS: 0 CONFIRMED: ...",
    "messageDetail": {
      "_id": "f3e108a9-eb75-4ec8-96c5-9dc2f5137d51-176",
      "mapping": "systemCsvfileAccounts_managedUser",
      "state": "SUCCESS",
      "stage": "COMPLETED_SUCCESS",
      "stageDescription": "reconciliation completed.",
      "progress": {
        "source": {
          "existing": {
            "processed": "2",
            "total": "2"
          }
        }
      }
    }
  }
}
```

Chapter 24

Synchronize Data Between MongoDB and IDM

This sample uses the Groovy Connector Toolkit to implement a scripted connector that interacts with a MongoDB database. You can use the connector to provision MongoDB database users and roles from an IDM managed repository.

The Groovy Connector Toolkit is bundled with IDM in the JAR `openidm/connectors/groovy-connector-1.5.20.8.jar`.

Sample Overview

The Groovy scripts required for the sample are bundled within the MongoDB connector. If you want to customize these scripts, you can specify different scripts by adjusting the `scriptRoots` property and script names in `provisioner.openicf-mongodb.json`.

This sample lets you to synchronize from IDM Managed User to an external MongoDB database.

Note

There is currently no way to synchronize passwords from an external MongoDB database to IDM. Because of this, it is recommended that IDM be used for user creation and password management.

While not demonstrated in this sample, the MongoDB connector can also:

- Synchronize from a dedicated store of IDM Managed MongoDB Roles to an external MongoDB database.
- Synchronize from an external MongoDB database to a dedicated IDM store of Managed MongoDB Roles.

Configure the MongoDB Database

This sample assumes a MongoDB database, running on the localhost system. Follow these steps to install and configure the MongoDB database:

1. Use the instructions for downloading and installing MongoDB in the [MongoDB Manual](#). For the supported version of MongoDB, see "MongoDB Connector" in the *Connectors Guide*.
2. Set up MongoDB, based on the `configurationProperties` discussed in "MongoDB Connector" in the *Connectors Guide*. By default, it listens on localhost, port 27017. For the purpose of this sample,

we'll set up an administrative user of `myUserAdmin` with a password of `Passw0rd` in the `admin` database. We will then create a database in MongoDB named `hrdb`.

Note

The MongoDB administrative user must have the `userAdminAnyDatabase` role, or attempts to update users will fail.

If want to use an existing MongoDB instance that runs on a different host or port, or you want to change the database credentials, adjust the `configurationProperties` in the connector configuration file (`samples/sync-with-mongodb/conf/provisioner.openicf-mongodb.json`) before you start the sample, as described in "Configuring the MongoDB Connector" in the *Connectors Guide*.

3. Set up the MongoDB database, with which IDM will synchronize its managed user repository, by:
 - Enabling authentication, as described in the following MongoDB document: *Enable Auth* .
 - Setting up users and roles, as described in this MongoDB document: *Manage Users and Roles* .

Run the Sample

In this section, you will start IDM with the sample configuration, test the connection to the MongoDB database, and populate the database with sample data.

The mapping configuration file (`sync.json`) for this sample includes one mapping: `managedUser_systemMongodbAccount`. You will use this mapping to synchronize users between the IDM repository and the MongoDB database:

1. Update `samples/sync-with-mongodb/conf/provisioner.openicf-mongodb.json` with the credentials and database information you created when configuring MongoDB. In our example, `database` would be set to `hrdb`, while `user` would be `myUserAdmin` with `userDatabase` set to `admin`.
2. Start IDM with the configuration for the MongoDB sample:

```
/path/to/openidm/startup.sh -p samples/sync-with-mongodb
```

3. Create at least one assignment and role to assign roles to users. In this example, we are creating a role to assign read privileges to users. The role created is conditional, and only assigned to active users:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-type: application/json" \
--request POST \
--data '{
  "name" : "MongoDB Read Access",
  "description": "Basic Read Access to HRDB",
  "mapping" : "managedUser_systemMongodbAccount",
  "attributes": [
```

```

    {
      "name": "roles",
      "value": [
        {
          "role": "read",
          "db": "hrdb"
        }
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ],
  \
"http://localhost:8080/openidm/managed/assignment?_action=create"
{
  "_id": "fb98f4a5-0f4d-4e22-9e17-79c45c11fe20",
  "_rev": "000000005c2da0eb",
  "name": "MongoDB Read Access",
  "description": "Basic Read Access to HRDB",
  "mapping": "managedUser_systemMongodbAccount",
  "attributes": [
    {
      "name": "roles",
      "value": [
        {
          "role": "read",
          "db": "hrdb"
        }
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ]
}

```

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "name": "MongoDB Read Access",
  "description": "Role for accounts with read access in MongoDB.",
  "condition": "/accountStatus eq \"active\"",
  "assignments": [
    {
      "_ref": "managed/assignment/fb98f4a5-0f4d-4e22-9e17-79c45c11fe20",
      "_refResourceCollection": "managed/assignment",
      "_refResourceId": "fb98f4a5-0f4d-4e22-9e17-79c45c11fe20"
    }
  ]
}' \
"http://localhost:8080/openidm/managed/role?_action=create"
{
  "_id": "5f16238e-39e1-4f8c-8b16-27d39dc64dc3",
  "_rev": "0000000011e566a2",
  "name": "MongoDB Read Access",
  "description": "Role for accounts with read access in MongoDB.",
  "condition": "/accountStatus eq \"active\""
}
```

4. Create new users in IDM. Note that MongoDB requires user name, password, and roles properties to successfully create a user. In this example, the **read** role is assigned to new users automatically.
5. Reconcile the managed user repository with the external MongoDB database:
 - To reconcile the repository through the UI:
 1. Log in to the Admin UI at the URL <https://localhost:8443/admin> as the default administrative user (**openidm-admin**) with password **openidm-admin**.
 2. Select Configure > Mappings.

The Mappings page shows one mapping: From the IDM Managed User repository to the MongoDB database (**managedUser_systemMongoDbAccount**).
 3. Select the **managedUser_systemMongoDbAccount** mapping, and choose the Reconcile option.
 - To reconcile the repository by using the command-line, launch the reconciliation operation with the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=managedUser_systemMongodbAccount&waitForCompletion=true"
{
  "_id": "e5bf074e-4da6-4ea7-8203-d4ec6f5a814a-24344",
  "state": "SUCCESS"
}
```

The reconciliation operation creates MongoDB users from the users found in `managed/user`.

Chapter 25

Synchronize Data Between IDM and HubSpot

This sample demonstrates bidirectional synchronization between IDM managed users and HubSpot contacts.

Prepare the Sample

The sample assumes that you have a client app in HubSpot, with the corresponding `clientID`, `clientSecret`, and `refreshToken`, and that you already have some contacts stored in HubSpot.

Get the Refresh Token

1. Point your browser to the following URL:

```
https://app.hubspot.com/oauth/authorize?client_id=clientID&scope=contacts&redirect_uri=your-domain
```

2. On the resulting page, select your user account in HubSpot. You are redirected to a URL similar to the following:

```
https://your-domain/?code=860c1867-9e4b-4761-82c4-1a5a4caf5224
```

3. Copy the `code` in this URL (`860...` in the preceding example). You will need the code for the following step.
4. Send the following POST request, substituting your `clientID`, `clientSecret`:

```
curl \
--request POST \
--header 'Cache-Control: no-cache' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data 'grant_type=authorization_code&client_id=your-id&client_secret=your-secret&redirect_uri=your-domain&code=code' \
"https://api.hubapi.com/oauth/v1/token"
{
  "refresh_token": "f37e1132-xxxx-xxxx-xxxx-xxxx",
  "access_token": "CKbm...VaE",
  "expires_in": 21600
}
```

Note that the `redirect_uri` must be URL-encoded in this request, for example, `redirect_uri=https%3A%2F%2Fwww.example.com%2F`.

The output of the POST request includes the `refresh_token` required to configure the connector.

Run the Sample

In this section, you will do the following:

1. Download and install the HubSpot connector.
2. Start IDM with the sample configuration.
3. Configure the HubSpot connector and test your connection to HubSpot.
4. Reconcile your HubSpot contacts with the IDM managed user repository.
5. Change a user in IDM and reconcile the changes back to HubSpot.

The mapping configuration file (`sync.json`) for this sample includes two mappings: `systemHubspotContact_managedUser` and `managedUser_systemHubspotContact`. You will use these mappings to reconcile users between IDM and HubSpot.

1. To install the HubSpot connector, download the connector jar from the ForgeRock BackStage download site and place it in the `/path/to/openidm/connectors` directory:

```
mv ~/Downloads/hubspot-connector-1.5.20.11.jar /path/to/openidm/connectors/
```

2. Start IDM with the configuration for the HubSpot sample:

```
/path/to/openidm/startup.sh -p samples/sync-with-hubspot
```

3. Configure the HubSpot connector.

You can do this by updating `samples/sync-with-hubspot/conf/provisioner.openicf-hubspot.json` with your HubSpot `clientId`, `clientSecret`, and `refreshToken`.

Alternatively, use the Admin UI to configure the connector. Select **Configure > Connectors**, select the HubSpot connector, and complete at least the Base Connector Details.

4. Test the connection to HubSpot by running the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=test"
{
  {
    "name": "hubspot",
    "enabled": true,
    "config": "config/provisioner.openicf/hubspot",
    "connectorRef": {
      "bundleVersion": "1.5.20.11",
      "bundleName": "org.forgerock.openicf.connectors.hubspot-connector",
      "connectorName": "org.forgerock.openicf.connectors.hubspot.HubspotConnector"
    },
    "displayName": "Hubspot Connector",
    "objectTypes": [
      "company",
      "contactProperties",
      "__ALL__",
      "companyProperties",
      "contact"
    ],
    "ok": true
  }
}
```

A status of `"ok": true` indicates that the connector can connect to HubSpot.

If you configure the connector through the Admin UI, the connection is tested as soon as you select Save.

5. Reconcile your HubSpot contacts with the IDM managed user repository by running the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?_action=recon&mapping=systemHubspotContact_managedUser&waitForCompletion=true"
{
  "_id": "1f148024-45b9-4dc1-9c3f-29976e02db00-3768",
  "state": "SUCCESS"
}
```

Alternatively, in the Admin UI, select Configure > Mappings, and select Reconcile on the `systemHubspotContact_managedUser` mapping.

6. In the Admin UI, select Manage > User and verify that your HubSpot contacts have been created as IDM managed users.
7. Edit one of the newly created managed users in IDM.

The easiest way to do this is to select Manage > User, select a user to edit, and change one of the user properties.

8. Reconcile the managed user repository with your HubSpot contacts by running the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=managedUser_systemHubspotContact&waitForCompletion=true"
{
  "_id": "1f148024-45b9-4dc1-9c3f-29976e02db00-8700",
  "state": "SUCCESS"
}
```

Alternatively, in the Admin UI, select Configure > Mappings, and select Reconcile on the `managedUser_systemHubspotContact` mapping.

9. Verify that the contact was updated correctly in HubSpot.

This is the end of the sample. For more information about the HubSpot connector, see "HubSpot Connector" in the *Connectors Guide*.

Chapter 26

Synchronize Data Between IDM and DocuSign

Important

This sample only works with DocuSign connector version 1.5.20.21 and lower. For more information, refer to the ICF documentation.

This sample demonstrates bidirectional synchronization between IDM managed users and DocuSign accounts.

The sample assumes that you have downloaded and installed the DocuSign connector and its dependencies, as described in "Install and Configure the DocuSign Connector" in the *Connectors Guide*. The sample also assumes that you have the DocuSign account information required to configure the connector, as described in "Before You Start" in the *Connectors Guide*.

Run the Sample

In this section, you will do the following:

1. Start IDM with the sample configuration.
2. Configure the DocuSign connector and test your connection to DocuSign.
3. Reconcile your DocuSign service accounts with the IDM managed user repository.
4. Change a user in IDM and reconcile the changes back to DocuSign.

The mapping configuration file (`sync.json`) for this sample includes two mappings: `systemDocuSignAccount_managedUser` and `managedUser_systemDocuSignAccount`. You will use these mappings to reconcile users between IDM and DocuSign.

1. Start IDM with the configuration for the DocuSign sample:

```
/path/to/openidm/startup.sh -p samples/sync-with-docusign
```

2. Configure the DocuSign connector.

You can configure the connector in two ways:

- Update `samples/sync-with-docusign/conf/provisioner.openicf-docusign.json` with your DocuSign account details.

- Use the Admin UI to configure the connector.

Follow one of the procedures in "Install and Configure the DocuSign Connector" in the *Connectors Guide* to configure the connector.

3. Test the connection to DocuSign by running the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=test"
{
  {
    "name": "docusign",
    "enabled": true,
    "config": "config/provisioner.openicf/docusign",
    "connectorRef": {
      "bundleVersion": "1.5.20.11",
      "bundleName": "org.forgerock.openicf.connectors.docusign-connector",
      "connectorName": "org.forgerock.openicf.connectors.docusign.DocuSignConnector"
    },
    "displayName": "DocuSign Connector",
    "objectTypes": [
      "userSignature",
      "signingGroup",
      "_ALL_",
      "account",
      "contact"
    ],
    "ok": true
  }
}
```

A status of `"ok": true` indicates that the connector can connect to DocuSign.

If you configure the connector through the Admin UI, the connection is tested as soon as you select Save.

4. Reconcile your existing DocuSign users with the IDM managed user repository by running the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?_action=recon&mapping=systemDocuSignAccount_managedUser&waitForCompletion=true"
{
  "_id": "7dac3ea9-c6be-4ff9-ae46-d8a0431949b3-7745",
  "state": "SUCCESS"
}
```

Alternatively, in the Admin UI, select Configure > Mappings, and select Reconcile on the `systemDocuSignAccount_managedUser` mapping.

5. In the Admin UI, select Manage > User and verify that your DocuSign users have been created as IDM managed users.
6. Edit one of the newly created managed users in IDM.

The easiest way to do this is to select Manage > User, select a user to edit, and change one of the user properties.

7. Reconcile the users in the managed user repository with your DocuSign users by running the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=managedUser_systemDocuSignAccount&waitForCompletion=true"
{
  "_id": "1f148024-45b9-4dc1-9c3f-29976e02db00-8700",
  "state": "SUCCESS"
}
```

Alternatively, in the Admin UI, select Configure > Mappings, and select Reconcile on the `managedUser_systemDocuSignAccount` mapping.

8. Verify that the contact was updated correctly in DocuSign.

This is the end of the sample. For more information about the DocuSign connector, see "DocuSign Connector" in the *Connectors Guide*.

Chapter 27

Synchronize Data Between IDM and a SCIM Provider

This sample demonstrates bidirectional synchronization between IDM and accounts configured to the *System for Cross-domain Identity Management*. As noted on their website, "The System for Cross-domain Identity Management (SCIM) specification is designed to make managing user identities in cloud-based applications and services easier."

While this sample has been built to comply with SCIM 2.0 standards, it's been tested with a SCIM 1.1 provider.

This sample assumes you've configured SCIM on a third-party system. From that system you'll need the following configuration properties:

- OAuth 2.0 Client ID
- OAuth 2.0 Client Secret
- OAuth 2.0 Token
- SCIM Endpoint
- SCIM Version
- Properties that you want to reconcile from the SCIM provider

Note

Depending on your provider, you may want to modify the `sync.json` file for this sample to match the properties from the SCIM provider to appropriate properties for IDM.

For more information on the SCIM connector, including properties for the `provisioner.openicf-scim.json` file, see "SCIM Connector" in the *Connectors Guide*.

Run the Sample

In this section, you will do the following:

- Start IDM with the sample configuration.

- Configure the SCIM connector and test your connection to the third-party SCIM provider.
- Reconcile your SCIM accounts with the IDM managed user repository.
- Change a user in IDM and reconcile the changes back to the third-party SCIM provider.
- Reconcile your SCIM roles with the IDM managed role repository.

The mapping configuration file (`sync.json`) for this sample includes four mappings, which you'll use to reconcile users and roles:

- `systemScimAccount_managedUser`
- `managedUser_systemScimAccount`
- `systemScimGroup_managedRole`
- `managedRole_systemScimGroup`

1. Start IDM with the configuration for the SCIM sample:

```
/path/to/openidm/startup.sh -p samples/sync-with-scim
```

2. Configure the SCIM connector, in the following configuration file: `samples/sync-with-scim/conf/provisioner.openicf-scim.json`.

Note

Depending on the requirements of your third-party SCIM provider, it may be acceptable to have a `null` value for properties such as `user`, `password`, and `tokenEndpoint`.

3. Test the connection to your third-party SCIM provider with the following command:


```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=test"
{
  {
    "name": "scim",
    "enabled": true,
    "config": "config/provisioner.openicf/scim",
    "connectorRef": {
      "bundleVersion": "1.5.20.12",
      "bundleName": "org.forgerock.openicf.connectors.scim-connector",
      "connectorName": "org.forgerock.openicf.connectors.scim.ScimConnector"
    },
    "displayName": "Scim Connector",
    "objectTypes": [
      "_ALL_",
      "account",
      "group"
    ],
    "ok": true
  }
}
```

A status of `"ok": true` indicates that the connector can connect to your third-party SCIM provider.

4. Reconcile your existing third-party SCIM users with the IDM managed user repository with the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemScimAccount_managedUser&waitForCompletion=true"
{
  "_id": "bdba3003-0c8a-4543-9efb-26269c78fa8b-96949",
  "state": "SUCCESS"
}
```

Alternatively, in the Admin UI, select **Configure > Mappings**, and select **Reconcile** on the `systemScimAccount_managedUser` mapping.

5. In the Admin UI, select **Manage > User** and verify that the users from the third-party SCIM provider have been created as IDM managed users.
6. Edit one of the newly created managed users in IDM.

The easiest way to do this is to select **Manage > User**, select a user to edit, and change one of the user properties.

7. Reconcile the users in the managed user repository with your SCIM users with the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=managedUser_systemScimAccount&waitForCompletion=true"
{
  "_id": "bdba3003-0c8a-4543-9efb-26269c78fa8b-104117",
  "state": "SUCCESS"
}
```

Alternatively, in the Admin UI, select Configure > Mappings, and select Reconcile on the `managedUser_systemScimAccount` mapping.

8. Verify that the contact was updated correctly on your third-party SCIM provider.
9. Repeat the process with roles. Reconcile existing third-party SCIM roles with IDM managed roles with the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemScimGroup_managedRole&waitForCompletion=true"
{
  "_id": "7dac3ea9-c6be-4ff9-ae46-d8a0431949b3-7745",
  "state": "SUCCESS"
}
```

Alternatively, in the Admin UI, select Configure > Mappings, and select Reconcile on the `systemScimGroup_managedRole` mapping.

10. Edit one of the newly created managed roles in IDM.

The easiest way to do this is to select Manage > Role, select a role to edit, and add a user to that role.

11. Reconcile the roles in the managed user repository with your SCIM users with the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=managedRole_systemScimGroup&waitForCompletion=true"
{
  "_id": "bdba3003-0c8a-4543-9efb-26269c78fa8b-112074",
  "state": "SUCCESS"
}
```

Alternatively, in the Admin UI, select **Configure > Mappings**, and select **Reconcile** on the **managedRole_systemScimGroup** mapping.

12. Verify that the role was updated correctly on your third-party SCIM provider.

Chapter 28

Subscribe to JMS Messages

IDM can subscribe to Java Messaging Service (JMS) messages using the Messaging Service's JMS Subscriber. In an event-driven architecture, also known as a message-driven architecture, there are publishers and subscribers. When a publisher sends a message over JMS, that message is broadcast. All active and subscribed clients receive that message. This sample shows how IDM can act as a JMS message subscriber, using the ActiveMQ JMS message broker.

Note

For more information on how IDM can publish JMS messages using the JMS Audit Event Handler, see "[Direct Audit Information to a JMS Broker](#)"

Sample Overview

With the scripted message handler shown in this sample, you can configure scripts to parse the contents of JMS messages, and act on that content.

The script in this sample, `crudpaqTextMessageHandler.js`, shows how JMS can handle ForgeRock REST operations. If you customize a script to manage JMS messages, you must also modify the `conf/messaging.json` file.

This sample uses ActiveMQ, a JMS message broker. With the ActiveMQ UI, you can act as the JMS message provider. This sample demonstrates how you can input REST payloads through the ActiveMQ UI.

Dependencies for JMS Messaging

The JMS audit event handler requires ActiveMQ, and a number of dependencies. This section lists the dependencies, where you can download them, and where they must be installed in the IDM instance. If you use a different ActiveMQ version, the dependency versions may differ from those shown.

Download the following files:

- ActiveMQ binary. This sample was tested with ActiveMQ 5.15.13.
- ActiveMQ Client that corresponds to your ActiveMQ version.

- JmDNS, version 3.4.1.
- The most recent **bnd** JAR file from <https://repo1.maven.org/maven2/biz/aQute/bnd/biz.aQute.bnd/>. The **bnd** utility lets you create OSGi bundles for libraries that do not yet support OSGi.
- Apache Geronimo J2EE management bundle.
- hawtbuf-1.11 JAR file (a Maven-based protocol buffer compiler).

1. Unpack the ActiveMQ binary. For example:

```
tar -zxvf ~/Downloads/apache-activemq-5.15.13-bin.tar.gz
```

2. Create a temporary directory, move the ActiveMQ Client, and **bnd** JAR files to that directory, then change to that directory:

```
mkdir ~/Downloads/tmp
mv activemq-client-5.15.13.jar ~/Downloads/tmp/
mv biz.aQute.bnd-version.jar ~/Downloads/tmp/
cd ~/Downloads/tmp/
```

3. Create an OSGi bundle as follows:

- a. In a text editor, create a BND file named **activemq.bnd** and save it to the current directory. The file should have the following contents:

```
version=5.15.13
Export-Package: *;version=${version}
Bundle-Name: ActiveMQ :: Client
Bundle-SymbolicName: org.apache.activemq
Bundle-Version: ${version}
```

Your **tmp/** directory should now contain the following files:

```
ls ~/Downloads/tmp/
activemq-client-5.15.13.jar activemq.bnd biz.aQute.bnd-version.jar
```

- b. In that same directory, create the OSGi bundle archive file as follows:

```
java -jar biz.aQute.bnd-version.jar \
wrap --properties activemq.bnd \
--output activemq-client-5.15.13-osgi.jar \
activemq-client-5.15.13.jar
```

4. Copy the resulting **activemq-client-5.15.13-osgi.jar** file to the **openidm/bundle** directory:

```
cp activemq-client-5.15.13-osgi.jar /path/to/openidm/bundle/
```

5. Copy the Apache Geronimo, hawtbuf, and JmDNS JAR files to the **openidm/bundle** directory:

```
cp ~/Downloads/geronimo-j2ee-management_1.1_spec-1.0.1.jar /path/to/openidm/bundle/
cp ~/Downloads/hawtbuf-1.11.jar /path/to/openidm/bundle/
cp ~/Downloads/jmdns-3.4.1.jar /path/to/openidm/bundle
```

Your IDM instance is now ready for you to configure the JMS audit event handler.

Configure SSL for ActiveMQ

This configuration provides a connection to the ActiveMQ server instance with TLSv1.3.

1. In the directory where you unpacked ActiveMQ, edit the `conf/activemq.xml` file as follows:

- In the `<brokers>` element, add an `<sslContext>`:

```
<broker xmlns="http://activemq.apache.org/schema/core" brokerName="localhost" dataDirectory="${activemq.data}">
  ...
  <sslContext>
    <sslContext keyStore="file:${activemq.conf}/broker.ks" keyStorePassword="password"/>
  </sslContext>
  ...
</broker>
```

- In the `<transportConnectors>` element, add an `ssl <transportConnector>`:

```
<transportConnectors>
  ...
  <transportConnector name="ssl" uri="ssl://0.0.0.0:61617?transport.needClientAuth=false"/>
</transportConnectors>
```

Note

To enable mutual authentication, set `transport.needClientAuth=true`, and import the IDM server certificate into the ActiveMQ truststore (`conf/broker.ts`).

2. Delete the existing self-signed server certificate from the ActiveMQ keystore and truststore:

```
keytool \
-delete \
-keystore /path/to/activeMQ/conf/broker.ts \
-alias broker-localhost
Enter keystore password: password
keytool \
-delete \
-keystore /path/to/activeMQ/conf/broker.ks \
-alias broker-localhost
Enter keystore password: password
```

3. Generate a new self-signed server certificate for ActiveMQ:

```
keytool \
-genkey \
-keyalg RSA \
-alias broker-localhost \
-keystore /path/to/activeMQ/conf/broker.ks \
-storepass password \
-validity 360 \
-keysize 2048
```

Important

The **CN** in the generated self-signed certificate *must* match the hostname that you specify in the IDM JMS provider URL. If you are using **localhost** to connect to the broker, you must specify **localhost** when **keytool** prompts you for the **first and last name**. If the **CN** is not the same as the hostname, the server certificate validation will fail.

4. Export the ActiveMQ server certificate:

```
keytool \  
-export \  
-alias broker-localhost \  
-file broker-localhost.key \  
-keystore /path/to/activeMQ/conf/broker.ks  
Enter keystore password: password  
Certificate stored in file <broker-localhost.key>
```

5. Import the ActiveMQ server certificate into the IDM truststore:

```
keytool \  
-import \  
-alias activemq \  
-keystore /path/to/openidm/security/truststore \  
-file broker-localhost.key  
Enter keystore password: changeit  
Owner: CN=localhost, OU=Unknown, O=example.com, L=Unknown, ST=Unknown, C=Unknown  
Issuer: CN=localhost, OU=Unknown, O=example.com, L=Unknown, ST=Unknown, C=Unknown  
...  
Trust this certificate? [no]: yes  
Certificate was added to keystore
```

Configure a Secure Port for JMS Messages

1. Edit `/path/to/openidm/samples/scripted-jms-subscriber/conf/audit.json`, as follows:

```
"java.naming.provider.url" : "ssl://localhost:61617?daemon=true"
```

2. Edit `/path/to/openidm/samples/scripted-jms-subscriber/conf/messaging.json`, as follows:

```
"java.naming.provider.url" : "ssl://localhost:61617?daemon=true"
```

Start the ActiveMQ Broker and IDM

With the appropriate bundles in the `/path/to/openidm/bundles` directory, you're ready to start the ActiveMQ message broker, as well as IDM with the JMS Audit Sample.

1. Navigate to the directory where you unpacked the ActiveMQ binary and run the following command to start the ActiveMQ broker:

```
cd ~/Downloads/apache-activemq-5.15.13
bin/activemq start
INFO: Loading '/path/to/Downloads/apache-activemq-5.15.13/bin/env'
INFO: Using java '/usr/bin/java'
INFO: Starting - inspect logfiles specified in logging.properties and log4j.properties to get details
INFO: pidfile created : '/path/to/Downloads/apache-activemq-5.15.13/data/activemq.pid' (pid '69627')
```

2. Start IDM, with the configuration for this sample:

```
cd /path/to/openidm/
./startup.sh -p samples/scripted-jms-subscriber
```

3. Go to the ActiveMQ Console at <http://localhost:8161/admin/topics.jsp>, and verify audit messages are being created and sent to the `forgerock.idm.audit` audit topic.

Access the REST Interface Using the ActiveMQ UI

In this section, you will run REST calls through the ActiveMQ UI. This assumes you started ActiveMQ and IDM in "Start the ActiveMQ Broker and IDM".

1. Access the ActiveMQ web console, at <http://localhost:8161/admin>. Log in as an administrator; the default administrative user and password are `admin` and `admin`.
2. In the ActiveMQ web console, in the Queue Name text box, enter the value of `idmQ` from the `conf/messaging.json` file. The default value for `queue.idmQ` and `destinationName` is `idmQ`. Enter that value and select Create.
3. In the Queues window for the `idmQ` queue, select the `Send To` link under Operations.
4. You should see a `Send a JMS Message` window, with a `Message body` text box towards the bottom.
5. Copy the following text, a payload to create a new user with a user ID of `mgr1`, to the `Message body` text box:

```
{
  "operation" : "CREATE",
  "resourceName" : "/managed/user",
  "newResourceId" : "mgr1",
  "content" : {
    "mail" : "mgr1@example.com",
    "sn" : "Sanchez",
    "givenName" : "Jane",
    "password" : "Password1",
    "employeenumber" : 100,
    "accountStatus" : "active",
    "roles" : [ ],
    "userName" : "mgr1"
  },
  "params" : {},
  "fields" : [ "*", "*_ref" ]
}
```

For comparison, note the following equivalent REST call to create the same user:


```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "mail" : "mgr1@example.com",
  "sn" : "Sanchez",
  "givenName" : "Jane",
  "password" : "Password1",
  "employeeNumber" : 100,
  "accountStatus" : "active",
  "roles" : [ ],
  "userName" : "mgr1",
  "params" : {},
  "fields" : [ "*", "*_ref" ]
}' \
"http://localhost:8080/openidm/managed/user?_action=create"
```

6. Observe the OSGi console. You should see output in two parts. The first part starts with:

```
*****request received*****
Parsed JMS JSON =
...
```

The first part should include a JSON-formatted replay of the request that you entered in the **Message body** text box.

The second part of the output includes information for that same user, as written to the managed user repository.

7. Confirm that user either in the Admin UI, or using the following REST call:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/mgr1"
```

8. Repeat the process to create additional users. Try a different REST function. For example, you can enter the following payload in the ActiveMQ UI **Message body** text box, to change the first name (**givenName**) of the **mgr1** user to Donna:

```
{
  "operation": "PATCH",
  "resourceName": "/managed/user/mgr1",
  "value": [
    {
      "operation": "replace",
      "field": "/givenName",
      "value": "Donna"
    }
  ]
}
```

Customize the Scripted JMS Sample

If you set up a custom script to parse and process JMS messages, store that script in the `script/` subdirectory. Assume that script is named `myCustomScript.js`.

Edit the `messaging.json` file in the `conf/` subdirectory, and point it to that file:

```
{
  "subscribers" : [
    {
      "name" : "IDM CREST Queue Subscriber",
      "instanceCount": 3,
      "enabled" : true,
      "type" : "JMS",
      "handler" : {
        "type" : "SCRIPTED",
        "properties" : {
          "script" : {
            "type" : "text/javascript",
            "file" : "myCustomScript.js"
          }
        }
      }
    },
    {
      "properties" : {
        "sessionMode" : "CLIENT",
        "jndi" : {
          "contextProperties" : {
            "java.naming.factory.initial" : "org.apache.activemq.jndi.ActiveMQInitialContextFactory",
            "java.naming.provider.url" : "tcp://127.0.0.1:61616?daemon=true",
            "queue.idmQ" : "idmQ"
          },
          "destinationName" : "idmQ",
          "connectionFactoryName" : "ConnectionFactory"
        }
      }
    }
  ]
}
```

You'll find some of these properties in "JMS Audit Event Handler Properties" in the *Audit Guide*. Despite the name of the table and the different configuration file, the properties are the same.

Other properties of interest in the `messaging.json` file are shown in the following table:

JMS `messaging.json` Configuration Properties

<code>messaging.json</code> File Label	Description
<code>subscribers</code>	Needed to subscribe to incoming JMS message requests
<code>name</code>	Arbitrary name for the subscriber
<code>instanceCount</code>	Each <code>instanceCount</code> manages a single connection between IDM and the messaging channel. Supports multithreading throughput. If subscribing to a queue,

messaging.json File Label	Description
	such as <code>queue.idmQ</code> , the message is handled by a single instance. If subscribing to a topic, all instances receive and handle the same message.
<code>handler</code>	Parses the JMS message, then processes it, possibly through a script
<code>queue.idmQ</code>	One of the JNDI context properties. Name of the JMS queue in the ActiveMQ UI.
<code>destinationName</code>	JNDI lookup name for message delivery

Chapter 29

Authenticate Using a Trusted Servlet Filter

This sample demonstrates how to use a custom servlet filter and the *Trusted Request Attribute Authentication Module* to allow IDM to authenticate through another service.

To configure authentication using ForgeRock Access Management, see the [Platform Setup Guide](#).

- "Prepare the Sample"
- "The Sample Servlet Filter"
- "Run the Sample"
- "Customize the Sample for an External System"

Prepare the Sample

Before you start this sample, complete the following steps:

- Prepare a fresh IDM installation, as described in "Prepare IDM".
- Download and install the [Apache Maven](#) build tool.
- Build the custom servlet filter bundle file:

```
cd /path/to/openidm/samples/trusted-servlet-filter/filter
mvn clean install
```

- Copy the newly built servlet bundle file to the `openidm/bundle` directory:

```
cp target/sample-trusted-servletfilter-1.0.jar /path/to/openidm/bundle
```

The Sample Servlet Filter

In the previous step, you built a bundle file from the Java file named `SampleTrustedServletFilter.java`, located in the directory `trustedServletfilter/filter/src/main/java/org/forgerock/openidm/sample/trustedServletfilter`.

The following line in that Java file looks for the `X-Special-Trusted-User` header, to identify a specific User ID as a "trusted" user.

```
final String specialHeader = ((HttpServletRequest) servletRequest).getHeader("X-Special-Trusted-User");
```

The next line sets the special Servlet attribute `X-ForgeRock-AuthenticationId` to this trusted User ID.

```
servletRequest.setAttribute("X-ForgeRock-AuthenticationId", specialHeader);
```

The rest of the servlet filter chain continues request processing:

```
filterChain.doFilter(servletRequest, servletResponse);
```

This sample includes a `servletfilter-trust.json` file that calls the compiled IDM trusted servlet `filterClass`:

```
{
  "classPathURLs" : [ ],
  "systemProperties" : { },
  "requestAttributes" : { },
  "scriptExtensions" : { },
  "initParams" : { },
  "urlPatterns" : [
    "/"
  ],
  "filterClass" : "org.forgerock.openidm.sample.trustedservletfilter.SampleTrustedServletFilter"
}
```

Run the Sample

In this section, you will demonstrate the servlet filter by configuring it with the special header described in the previous section. Normally, a servlet filter used for authentication does not allow a client to masquerade as any user. This sample demonstrates a basic use of a servlet filter by establishing the authentication ID.

1. Start IDM with the configuration for the trusted filter sample:

```
/path/to/openidm/startup.sh -p samples/trusted-servlet-filter
```

2. Create a new managed user, Barbara Jensen, with `userName` `bjensen`:

```
curl \
--header "X-OpenIDM-Password: openidm-admin" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '
{
  "userName": "bjensen",
  "telephoneNumber": "6669876987",
  "givenName": "Barbara",
  "sn": "Jensen",
  "description": "Example User",
}
```

```

    "mail": "bjensen@example.com",
    "authzRoles" : [
      {
        "_ref" : "internal/role/openidm-authorized"
      }
    ]
  }' \
"http://localhost:8080/openidm/managed/user"
{
  "_id": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
  "_rev": "000000004988917b",
  "userName": "bjensen",
  "telephoneNumber": "6669876987",
  "givenName": "Barbara",
  "sn": "Jensen",
  "description": "Example User",
  "mail": "bjensen@example.com",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}

```

Note the ID of the new user—you will need it in the steps that follow.

3. Use the special request header **X-Special-Trusted-User** to authenticate bjensen (specifying her ID as the header value).

```

curl \
--header "X-Special-Trusted-User: 9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/info/login?_fields=authenticationId,authorization"
{
  "_id": "login",
  "authenticationId": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
  "authorization": {
    "component": "managed/user",
    "roles": [
      "internal/role/openidm-authorized"
    ],
    "id": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
    "moduleId": "TRUSTED_ATTRIBUTE"
  }
}

```

Note that the output includes the user's authentication and authorization details. In this case, bjensen, with ID **9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb** is authenticated with the **openidm-authorized** role.

Customize the Sample for an External System

To customize this sample for an external authentication/authorization system, you need a servlet filter which authenticates against that external system. You may use a third-party supplied filter, or develop your own filter, using the one in this sample as a model.

The filter you use should have at least the following capabilities:

- Perform REST calls to another system.
- Search through databases.
- Inspect headers related to authentication and authorization requests.

This servlet filter must set the username of the authenticated user in a special request attribute. You need to configure that same attribute name in the `TRUSTED_ATTRIBUTE` authentication module, specifically the value of `authenticationIdAttribute`.

It is helpful if you have a filter that returns an object with the `userRoles` property. If your filter does not support queries using the following parameter:

```
queryOnResource + "/" + authenticationId
```

You will need to provide a security context augmentation script that populates the following authorization properties in the "security" object:

- `security.authorization.component`
- `security.authorization.roles`

The value for the `security.authorization.component` is automatically set to the value specified in any existing `queryOnResource` property.

Chapter 30

Create a Custom Endpoint

Scriptable custom endpoints let you launch arbitrary scripts using the IDM REST URI. For information about how custom endpoints are configured, see "*Create Custom Endpoints to Launch Scripts*" in the *Scripting Guide*.

The example endpoint provided in `/path/to/openidm/samples/example-configurations/custom-endpoint` illustrates the configuration of a custom endpoint, and the structure of custom endpoint scripts.

The purpose of this custom endpoint is to return a list of variables available to each method used in a script. The scripts show the complete set of methods that can be used. These methods map to the standard HTTP verbs - create, read, update, delete, patch, query, and action. A sample JavaScript and Groovy script is provided.

To run the sample:

1. Copy the endpoint configuration file (`samples/example-configurations/custom-endpoint/conf/endpoint-echo.json`) to your project's `conf` directory.
2. Copy either the JavaScript file (`samples/example-configurations/custom-endpoint/script/echo.js`) or Groovy script file (`samples/example-configurations/custom-endpoint/script/echo.groovy`) to your project's `script` directory.
3. Open the endpoint configuration file in a text editor:

```
{
  "file" : "echo.groovy",
  "type" : "groovy",
  "_file" : "echo.js",
  "_type" : "text/javascript",
  ...
}
```

The configuration file contains a reference to the endpoint scripts. In this case, the JavaScript script is commented out (with an underscore before the `file` and `type` properties). If you want to use the JavaScript endpoint script, uncomment these lines and comment out the lines that correspond to the Groovy script in the same way.

Endpoint configuration files can include a `context` property that specifies the route to the endpoint, for example:

```
"context" : "endpoint/linkedView/*"
```

If no `context` is specified, the route to the endpoint is taken from the file name, in this case `endpoint/echo`.

4. Test each method in succession to return the expected request structure of that method. The following examples show the request structure of the read, create and patch methods. The configuration file has been edited to use the JavaScript file, rather than the Groovy file. The output shown in these examples has been cropped for legibility. For a description of each parameter, see "Custom Endpoint Scripts" in the *Scripting Guide*.

The following command performs a read on the echo endpoint and returns the request structure of a read request:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/endpoint/echo"
{
  "_id": "",
  "method": "read",
  "context": {
    "class": "org.forgerock.http.routing.ApiVersionRouterContext",
    "name": "apiVersionRouter",
    "defaultVersionBehaviour": "LATEST",
    "warningEnabled": false,
    "resourceVersion": "1.0",
    "parent": {
      "class": "org.forgerock.http.routing.UriRouterContext",
      "name": "router",
      ...
    }
  },
  "resourceName": "",
  "parameters": {}
}
```

The following command performs a query on the echo endpoint and returns the request structure of a query request:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/endpoint/echo?_queryFilter=true"
{
  "result": [
    {
      "method": "query",
      "pageSize": 0,
      "queryFilter": "true",
      "resourceName": "",
      "pagedResultsOffset": 0,
      "pagedResultsCookie": null,
      "parameters": {},
      "content": null,
      "queryId": null,
      "content": null,
      "context": {
        ...
      }
    }
  ],
  ...
}
```

The following command sends a create request to the echo endpoint. No user is actually created. The endpoint script merely returns the request structure of a create request. The **content** parameter in this case provides the JSON object that was sent with the request:

```
curl \
--header "X-OpenIDM-Password: openidm-admin" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--data '{
  "userName": "steve",
  "givenName": "Steve",
  "sn": "Carter",
  "telephoneNumber": "0828290289",
  "mail": "scarter@example.com",
  "password": "Passw0rd"
}' \
--request POST \
"http://localhost:8080/openidm/endpoint/echo?_action=create"
{
  "_id": "",
  "method": "create",
  "resourceName": "",
  "newResourceId": null,
  "parameters": {},
  "content": {
    "userName": "steve",
    "givenName": "Steve",
    "sn": "Carter",
    "telephoneNumber": "0828290289",

```

```

    "mail": "scarter@example.com",
    "password": "Passw0rd"
  },
  "context": {
    ...
  }
}

```

The following command sends a patch request to the echo endpoint.

```

curl \
--header "X-OpenIDM-Password: openidm-admin" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--data '[
  {
    "operation": "replace",
    "field": "/givenName",
    "value": "Steven"
  }
]' \
--request PATCH \
"http://localhost:8080/openidm/endpoint/echo"
{
  "_id": "",
  "method": "patch",
  "resourceName": "",
  "revision": null,
  "patch": [
    {
      "operation": "replace",
      "field": "/givenName",
      "value": "Steven"
    }
  ],
  "parameters": {},
  "context": {
    ...
  }
}

```

IDM Glossary

correlation query	A correlation query specifies an expression that matches existing entries in a source repository to one or more entries in a target repository. A correlation query might be built with a script, but it is not the same as a correlation script. For more information, see <i>"Correlating Source Objects With Existing Target Objects"</i> in the <i>Synchronization Guide</i> .
correlation script	A correlation script matches existing entries in a source repository, and returns the IDs of one or more matching entries on a target repository. While it skips the intermediate step associated with a correlation query , a correlation script can be relatively complex, based on the operations of the script.
entitlement	An entitlement is a collection of attributes that can be added to a user entry via roles. As such, it is a specialized type of assignment . A user or device with an entitlement gets access rights to specified resources. An entitlement is a property of a managed object.
JCE	Java Cryptographic Extension, which is part of the Java Cryptography Architecture, provides a framework for encryption, key generation, and digital signatures.
JSON	JavaScript Object Notation, a lightweight data interchange format based on a subset of JavaScript syntax. For more information, see the JSON site .
JSON Pointer	A JSON Pointer defines a string syntax for identifying a specific value within a JSON document. For information about JSON Pointer syntax, see the JSON Pointer RFC .

JWT	JSON Web Token. As noted in the JSON Web Token draft IETF Memo , "JSON Web Token (JWT) is a compact URL-safe means of representing claims to be transferred between two parties." For IDM, the JWT is associated with the JWT_SESSION authentication module.
managed object	An object that represents the identity-related data managed by IDM. Managed objects are configurable, JSON-based data structures that IDM stores in its pluggable repository. The default configuration of a managed object is that of a user, but you can define any kind of managed object, for example, groups or roles.
mapping	A policy that is defined between a source object and a target object during reconciliation or synchronization. A mapping can also define a trigger for validation, customization, filtering, and transformation of source and target objects.
OSGi	A module system and service platform for the Java programming language that implements a complete and dynamic component model. For more information, see What is OSGi? Currently, only the Apache Felix container is supported.
reconciliation	During reconciliation, comparisons are made between managed objects and objects on source or target systems. Reconciliation can result in one or more specified actions, including, but not limited to, synchronization.
resource	An external system, database, directory server, or other source of identity data to be managed and audited by the identity management system.
REST	Representational State Transfer. A software architecture style for exposing resources, using the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed.
role	IDM distinguishes between two distinct role types - provisioning roles and authorization roles. For more information, see "Managed Roles" in the <i>Object Modeling Guide</i> .
source object	In the context of reconciliation, a source object is a data object on the source system, that IDM scans before attempting to find a corresponding object on the target system. Depending on the defined mapping, IDM then adjusts the object on the target system (target object).
synchronization	The synchronization process creates, updates, or deletes objects on a target system, based on the defined mappings from the source system. Synchronization can be scheduled or on demand.

system object	A pluggable representation of an object on an external system. For example, a user entry that is stored in an external LDAP directory is represented as a system object in IDM for the period during which IDM requires access to that entry. System objects follow the same RESTful resource-based design principles as managed objects.
target object	In the context of reconciliation, a target object is a data object on the target system, that IDM scans after locating its corresponding object on the source system. Depending on the defined mapping, IDM then adjusts the target object to match the corresponding source object.