



Installation Guide

/ OpenIDM 3.1

Latest update: 3.1.0

Mark Craig
Lana Frost
Paul Bryan
Andi Egloff
Laszlo Hordos
Matthias Tristl
Mike Jang

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

Copyright © 2011-2017 ForgeRock AS.

Abstract

Guide to installing and evaluating OpenIDM. The OpenIDM project offers flexible, open source services for automating management of the identity life cycle.



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, <http://fontawesome.io>.

This Font Software is licensed under the SIL Open Font License, Version 1.1. This license is available with a FAQ at: <http://scripts.sil.org/OFL>.

Table of Contents

Preface	v
1. Who Should Use this Guide	v
2. Formatting Conventions	v
3. Accessing Documentation Online	vi
4. Using the ForgeRock.org Site	vi
1. Installing OpenIDM Services	1
1.1. Before You Run OpenIDM	1
1.2. Installing and Running OpenIDM	2
1.3. To Get Started With the OpenIDM REST Interface	5
1.4. OpenIDM User Interfaces	10
2. First OpenIDM Sample	11
2.1. Before You Begin	11
2.2. About the Sample	12
2.3. Install the Sample	16
2.4. Review the Sample in the Administrative User Interface	16
2.5. Running Reconciliation	17
2.6. Viewing Users and Logs	19
2.7. Adding Users in a Resource	24
2.8. Adding Users Over REST	26
3. More OpenIDM Samples	29
3.1. Before You Begin	29
3.2. Sample 1 - XML File	30
3.3. Sample 2 - LDAP One Way	30
3.4. Sample 2b - LDAP Two Way	36
3.5. Sample 2c - Synchronizing LDAP Group Membership	44
3.6. Sample 2d - Synchronizing LDAP Groups	53
3.7. Using the Groovy Connector Toolkit to Create Scripted Connectors	57
3.8. Sample - Using the PowerShell Connector Toolkit to Create Scripted Connectors	92
3.9. Sample 4 - CSV to XML File	101
3.10. Sample 5 - Synchronization of Two Resources	102
3.11. Sample 5b - Failure Compensation With Multiple Resources	105
3.12. Sample 6 - LiveSync With an AD Server	107
3.13. Sample 7 - Scripting a SCIM-like Schema	115
3.14. Sample 8 - Logging in Scripts	117
3.15. Sample 9 - Asynchronous Reconciliation Using Workflows	118
3.16. Sample - Configuring Authentication Management With OpenAM	121
3.17. Sample - Demonstrate Extended Audit Capabilities	126
3.18. Sample - Connecting to Google With the Google Apps Connector	129
3.19. Sample - Connecting to Salesforce With the Salesforce Connector	136
4. Installing a Repository For Production	151
4.1. To Set Up OpenIDM With MySQL	151
4.2. To Set Up OpenIDM With MS SQL	154
4.3. To Set Up OpenIDM With Oracle Database	158

4.4. To Set Up OpenIDM With PostgreSQL	162
5. Removing and Moving OpenIDM Software	164
6. Migrating to OpenIDM 3.1.0	165
OpenIDM Glossary	169
Index	171

Preface

This guide shows you how to install core OpenIDM services for identity management, provisioning, and compliance. Unless you are planning a throwaway evaluation or test installation, read the *Release Notes* before you get started.

1. Who Should Use this Guide

This guide is written for anyone installing OpenIDM to manage and to provision identities, and to ensure compliance with identity management regulations.

This guide covers the install and removal (uninstall) procedures that you theoretically perform only once per version. This guide aims to provide you with at least some idea of what happens behind the scenes when you perform the steps.

This guide also takes you through all of the samples provided with OpenIDM.

You do not need to be an OpenIDM wizard to learn something from this guide, though 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 application servers. You can nevertheless get started with this guide, and then learn more as you go along.

If you have a previous version of OpenIDM installed, see the *Compatibility* in the *Release Notes* section of the *Release Notes* before installing this version.

2. Formatting Conventions

Most examples in the documentation are created in GNU/Linux or Mac OS X operating environments. If distinctions are necessary between operating environments, examples are labeled with the operating environment name in parentheses. To avoid repetition file system directory names are often given only in UNIX format as in `/path/to/server`, even if the text applies to `C:\path\to\server` as well.

Absolute path names usually begin with the placeholder `/path/to/`. This path might translate to `/opt/`, `C:\Program Files\`, or somewhere else on your system.

Command-line, terminal sessions are formatted as follows:

```
$ echo $JAVA_HOME
/path/to/jdk
```

Command output is sometimes formatted for narrower, more readable output even though formatting parameters are not shown in the command.

Program listings are formatted as follows:

```
class Test {
    public static void main(String [] args) {
        System.out.println("This is a program listing.");
    }
}
```

3. Accessing Documentation Online

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

4. Using the ForgeRock.org Site

The [ForgeRock.org](https://forgerock.org) site has links to source code for ForgeRock open source software, as well as links to the ForgeRock forums and technical blogs.

If you are a *ForgeRock customer*, raise a support ticket instead of using the forums. ForgeRock support professionals will get in touch to help you.

Chapter 1

Installing OpenIDM Services

This chapter covers the tasks required to install and start OpenIDM.

1.1. Before You Run OpenIDM

This section covers what you need to know before running OpenIDM.

1.1.1. Java Environment

This release of OpenIDM requires Java Development Kit 6 or Java Development Kit 7. ForgeRock recommends the most recent update of Java 6 or 7 to ensure that you have the latest security fixes.

In addition, on Windows systems, you must set the `JAVA_HOME` environment variable to point to the root of a valid Java installation. The following steps indicate how to set the `JAVA_HOME` environment variable on Windows Server 2008 R2. Adjust the steps for your specific environment.

- Locate your JRE Installation Directory. If you have not changed the installation path for the Java Runtime Environment during installation, it will be in a directory under `C:\Program Files\Java\`.
- Select Start > Control Panel > System and Security > System.
- Click Advanced System Settings.
- Click Environment Variables.
- Under System Variables, click New.
- Enter the Variable name (`JAVA_HOME`) and set the Variable value to the JRE installation directory, for example `C:\Program Files\Java\jre7`.
- Click OK.

1.1.2. Application Container

OpenIDM services run in an OSGi container with an embedded Servlet container, and an embedded noSQL database. By default the OSGi container is Apache Felix. The default Servlet container is Jetty. For OpenIDM 3.1, the only supported configuration is running the services in Apache Felix and Jetty.

1.2. Installing and Running OpenIDM

Follow the procedures in this section to install and run OpenIDM.

Procedure 1.1. To Install OpenIDM Services

Follow these steps to install OpenIDM.

1. Make sure you have an appropriate version of Java installed.

```
$ java -version  
  
java version "1.6.0_24"  
Java(TM) SE Runtime Environment (build 1.6.0_24-b07-334)  
Java HotSpot(TM) 64-Bit Server VM (build 19.1-b02-334, mixed mode)
```

Check the release notes for Java requirements in the chapter, *Before You Install OpenIDM Software* in the *Release Notes*.

2. Download OpenIDM from one of the following locations:

- The [Forgerock.com Download](#) page has the latest stable, supported release of OpenIDM Enterprise, along with the other products in the ForgeRock Identity Platform.
- [Builds](#) includes the nightly build and the OpenIDM agents. Note that this is the working version of the trunk and should not be used in a production environment.
- [Archives](#) includes the stable builds for all previous releases of OpenIDM.

3. Unpack the contents of the .zip file into the install location.

```
$ cd /path/to  
$ unzip ~/Downloads/openidm-3.1.zip  
  
...  
inflating: openidm/connectors/scriptedsql-connector-1.4.1.0.jar  
inflating: openidm/bin/felix.jar  
inflating: openidm/bin/openidm.jar
```

4. (Optional) By default, OpenIDM listens for HTTP and HTTPS connections on ports 8080 and 8443, respectively. To change the default port, edit the `/path/to/openidm/conf/boot/boot.properties` file. For more information, see [Appendix B, "Ports Used"](#) in the *Integrator's Guide*.
5. Before running OpenIDM in production, replace the default OrientDB repository provided for evaluation with a JDBC repository.

For more information, see [Chapter 4, "Installing a Repository For Production"](#).

Procedure 1.2. To Start OpenIDM Services

Follow these steps to run OpenIDM interactively.

To run OpenIDM as a background process, see Chapter 2, "Starting and Stopping OpenIDM" in the *Integrator's Guide*.

1. Start the Felix container, load all OpenIDM services, and start a command shell to allow you to manage the container.
 - Start OpenIDM (UNIX).

```

$ ./startup.sh

Using OPENIDM_HOME:  /path/to/openidm
Using PROJECT_HOME:  /path/to/openidm
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/conf/boot/boot.properties
OpenIDM version "3.1" (revision:
  XXXX)
-> OpenIDM ready
    
```

- Start OpenIDM (Windows).

```

C:\> cd \path\to\openidm
C:\> startup.bat

"Using OPENIDM_HOME:  \path\to\openidm"
"Using PROJECT_HOME:  \path\to\openidm"
"Using OPENIDM_OPTS:  -Xmx1024m -Xms1024m -Dfile.encoding=UTF-8"
"Using LOGGING_CONFIG: -Djava.util.logging.config.file=\path\to\openidm\conf\logging.properties"
Using boot properties at \path\to\openidm\conf\boot\boot.properties
OpenIDM version "3.1" (revision:
  XXXX)
-> OpenIDM
  ready
->
    
```

At the resulting `->` prompt, you can enter commands such as **help** for usage, or **ps** to view the bundles installed. To see a list of all the OpenIDM core services and their states, enter the following command.

```

-> scr list

  Id  State      Name
[ 12] [active   ] org.forgerock.openidm.endpoint
[ 13] [active   ] org.forgerock.openidm.endpoint
[ 14] [active   ] org.forgerock.openidm.endpoint
[ 15] [active   ] org.forgerock.openidm.endpoint
[ 16] [active   ] org.forgerock.openidm.endpoint
[ 17] [active   ] org.forgerock.openidm.endpoint
[ 23] [unsatisfied] org.forgerock.openidm.info
[ 27] [active   ] org.forgerock.openidm.provisioner.openicf.connectorinfoprovider
    
```

```
[ 35] [active      ] org.forgerock.openidm.ui.simple
[ 29] [active      ] org.forgerock.openidm.restlet
[  3] [active      ] org.forgerock.openidm.repo.orientdb
[  7] [active      ] org.forgerock.openidm.scope
[  5] [active      ] org.forgerock.openidm.audit
[ 32] [active      ] org.forgerock.openidm.schedule
[  2] [unsatisfied] org.forgerock.openidm.repo.jdbc
[ 31] [active      ] org.forgerock.openidm.workflow
[  9] [active      ] org.forgerock.openidm.managed
[ 28] [active      ] org.forgerock.openidm.provisioner.openicf
[ 22] [active      ] org.forgerock.openidm.health
[ 26] [active      ] org.forgerock.openidm.provisioner
[  0] [active      ] org.forgerock.openidm.config.starter
[ 34] [active      ] org.forgerock.openidm.taskscanner
[ 20] [active      ] org.forgerock.openidm.external.rest
[  6] [active      ] org.forgerock.openidm.router
[ 33] [active      ] org.forgerock.openidm.scheduler
[ 19] [unsatisfied] org.forgerock.openidm.external.email
[ 11] [active      ] org.forgerock.openidm.sync
[ 25] [active      ] org.forgerock.openidm.policy
[  8] [active      ] org.forgerock.openidm.script
[ 10] [active      ] org.forgerock.openidm.recon
[  4] [active      ] org.forgerock.openidm.http.contextregistrator
[  1] [active      ] org.forgerock.openidm.config
[ 18] [active      ] org.forgerock.openidm.endpointservice
[ 30] [unsatisfied] org.forgerock.openidm.servletfilter
[ 24] [active      ] org.forgerock.openidm.infoservice
[ 21] [active      ] org.forgerock.openidm
.authentication
->
```

A default startup does not include certain configurable services, which will indicate an **unsatisfied** state until they are included in the configuration. As you work through the sample configurations described later in this guide, you will notice that these services are active.

Startup errors and messages are logged to the console by default. You can also view these messages in the log files at `/path/to/openidm/logs`.

2. Alternatively, you can manage the container and services from the Felix administration console.

Use these hints to connect to the console.

- Default Console URL: `https://localhost:8443/system/console`
- Default user name: `admin`
- Default password: `admin`

Some basic hints on using the Felix administration console follow.

- Select the Components tab to see OpenIDM core services and their respective states.
- Select the Shell tab to access the `->` prompt.

- Select the System Information tab to stop or restart the container.

Procedure 1.3. To Stop the OpenIDM Services

- You can stop OpenIDM Services from the `->` prompt, or through the Felix console.
- Either enter the **shutdown** command at the `->` prompt.

```
-> shutdown
...
$
```

- Or click Stop on the System Information tab of the Felix console, by default `http://localhost:8080/system/console`.

This stops the Servlet container as well, and the console is no longer accessible.

- On Unix systems, you can stop OpenIDM by using the **shutdown.sh** script, located in the `/path/to/openidm` directory.

```
$ ./shutdown.sh
./shutdown.sh
Stopping OpenIDM (31391)
```

1.3. To Get Started With the OpenIDM REST Interface

OpenIDM provides RESTful access to users in the OpenIDM repository. To access the OpenIDM repository over REST, you can use a browser-based REST client, such as the *Simple REST Client* for Chrome, or *RESTClient* for Firefox. Alternatively you can use the **curl** command-line utility that is included with most operating systems. For more information about **curl**, see <https://github.com/bagder/curl>.

OpenIDM is accessible over the regular and secure HTTP ports of the Jetty Servlet container, 8080 and 8443.

If you want to run **curl** over the secure port, 8443, you must either include the **--insecure** option, or follow the instructions shown in *Restrict REST Access to the HTTPS Port* in the *Integrator's Guide*. You can use those instructions with the self-signed certificate that is generated when OpenIDM starts, or with a `*.cert` file provided by a certificate authority.

In numerous cases, **curl** commands to the secure port are depicted with a `--cacert self-signed.crt` option. Instructions for creating that `self-signed.crt` file are shown in the aforementioned section on *Restrict REST Access to the HTTPS Port* in the *Integrator's Guide*.

If you would rather use **curl** to connect to the regular HTTP port, omit the `--cacert self-signed.crt` file and point to a regular Jetty HTTP URL such as `http://localhost:8080/openidm/...`

Note

All RESTful command line examples in this guide, as depicted with **curl**, are based on the default configuration of OpenIDM. If you change configuration files in directories such as `openidm/conf` and `openidm/script`, you might need to modify the RESTful commands to reflect those changes.

Most of the examples in this guide use client-assigned IDs when creating resources, as it makes the examples easier to read.

In general, server-assigned UUIDs are better in production, as they can be generated easily in clustered environments.

1. Access the following URL to obtain the JSON representation of all users in the OpenIDM repository.

```
$ curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
http://localhost:8080/openidm/managed/user/?_queryId=query-all-ids
```

When you first install OpenIDM with an empty repository, no users exist.

2. Create a user `joe` by sending a RESTful POST.

The following **curl** commands create the user `joe` in the repository.

- Create `joe` (UNIX).

```
$ curl \
--cacert self-signed.crt \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request POST \
--data '{ \
  "userName": "joe", \
  "givenName": "joe", \
  "sn": "smith", \
  "mail": "joe@example.com", \
  "telephoneNumber": "555-123-1234", \
  "password": "TestPassw0rd", \
  "description": "My first user", \
  "_id": "joe" \
}' \
https://localhost:8443/openidm/managed/user?_action=create

{
  "userName": "joe",
  "stateProvince": "",
  "postalAddress": "",
  "effectiveAssignments": {},
  "roles": [
    "openidm-authorized"
  ],
  "telephoneNumber": "555-123-1234",
  "password": {
```

```

"$crypto": {
  "value": {
    "key": "openidm-sym-default",
    "iv": "gTcveNaZdSHE1qeBgcmzRw==",
    "cipher": "AES/CBC/PKCS5Padding",
    "data": "X9sCuuvNwSbblxdqS65qxw=="
  },
  "type": "x-simple-encryption",
}
},
"effectiveRoles": [
  "openidm-authorized"
],
"givenName": "joe",
"address2": "",
"lastPasswordAttempt": "Tue Feb 25 2014 18:03:40 GMT-0800 (PST)",
"passwordAttempts": "0",
"sn": "smith",
"mail": "joe@example.com",
"country": "",
"city": "",
"rev": "1",
"lastPasswordSet": "",
"postalCode": "",
"accountStatus": "active",
"description": "My first user",
"_id": "joe"
}

```

- Create **joe** (Windows).

```

C:\> curl ^
--cacert self-signed.crt ^
--header "Content-Type: application/json" ^
--header "X-OpenIDM-Username: openidm-admin" ^
--header "X-OpenIDM-Password: openidm-admin" ^
--request POST ^
--data "{ ^
\"userName\": \"joe\", ^
\"givenName\": \"joe\", ^
\"sn\": \"smith\", ^
\"mail\": \"joe@example.com\", ^
\"telephoneNumber\": \"555-123-1234\", ^
\"password\": \"TestPassw0rd\", ^
\"description\": \"My first user\" ^
\"_id\": \"joe\" ^
}" ^
https://localhost:8443/openidm/managed/user?_action=create

```

3. Fetch the newly created user from the repository with a RESTful GET.

```

$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
https://localhost:8443/openidm/managed/user/joe

```

```
{
  "effectiveAssignments": {},
  "effectiveRoles": [
    "openidm-authorized"
  ],
  "stateProvince": "",
  "userName": "joe",
  "postalAddress": "",
  "givenName": "joe",
  "address2": "",
  "lastPasswordAttempt": "Tue Feb 25 2014 18:13:03 GMT-0800 (PST)",
  "passwordAttempts": "0",
  "sn": "smith",
  "mail": "joe@example.com",
  "_rev": "1",
  "_id": "joe",
  "country": "",
  "city": "",
  "lastPasswordSet": "",
  "postalCode": "",
  "description": "My first user",
  "accountStatus": "active",
  "telephoneNumber": "555-123-1234",
  "roles": [
    "openidm-authorized"
  ]
}
```

4. Notice that more attributes are returned for user `joe` than the attributes you added in the previous step. The additional attributes are added by a script named `onCreate-user-set-default-fields.js` that is triggered when a new user is created. For more information, see Section C.1.5, "Managed Object Configuration" in the *Integrator's Guide*.

When you create a user some attributes might be required by the policy associated with that user. These are listed in the `conf/policy.json` file.

1.3.1. Format REST Output for Readability

With all `curl`-based REST calls, OpenIDM returns the JSON object all on one line.

Without a bit of help, the JSON output is formatted all on one line. One example is shown below, and it is difficult to read:

```
{
  "mail": "joe@example.com",
  "sn": "smith",
  "passwordAttempts": "0",
  "lastPasswordAttempt": "Mon Apr 14 2014 11:13:37 GMT-0800 (GMT-08:00)",
  "address2": "",
  "givenName": "joe",
  "effectiveRoles": ["openidm-authorized"],
  "password": {
    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "data": "0BFVL9cG8uaLoolN+SMJ3g==",
        "cipher": "AES/CBC/PKCS5Padding",
        "iv": "7r1V4EkwDRHkt19F8g22A==",
        "key": "openidm-sym-default"
      }
    }
  },
  "country": "",
  "city": "",
  "_rev": "1",
  "lastPasswordSet": "",
  "postalCode": "",
  "_id": "joe3",
  "description": "My first user",
  "accountStatus": "active",
  "telephoneNumber": "555-123-1234",
  "roles": ["openidm-authorized"],
  "effectiveAssignments": {},
  "postalAddress": "",
  "stateProvince": "",
  "userName": "joe3"
}
```

At least two options are available to clean up this output.

The standard way to format JSON output is with a JSON parser such as `jq`. You would "pipe" the output of a REST call to `jq`, as follows:

```
$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --request GET \
  "https://localhost:8443/openidm/managed/user/joe" \
  | jq .
```

The ForgeRock REST API includes an optional `_prettyPrint` request parameter. The default value is `false`. To use the ForgeRock REST API to format output, add a parameter such as `?_prettyPrint=true` or `&_prettyPrint=true`, depending on whether it is added to the end of an existing request parameter. In this case, the following command would return formatted output:

```
$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --request GET \
  "https://localhost:8443/openidm/managed/user/joe?_prettyPrint=true"
```

Note that most command-line examples in this guide do not show this parameter, although the output is formatted for readability.

1.4. OpenIDM User Interfaces

OpenIDM supports configuration from Web-based user interfaces (WUI), hereafter known as the UI.

OpenIDM includes UIs at two different endpoints, `/openidmui` and `/admin`. We refer to the administrative tools available at each endpoint as the User View UI and the Administrative UI (Admin UI), respectively.

The User View UI allows regular users to update parts of their own profiles, such as password updates and user data such as addresses. When enabled, anonymous users can self-register and reset their passwords, as described in the following section: *Configuring the Default User Interface* in the *Integrator's Guide*.

Administrative users can do more in the User View UI. They can create and delete user accounts. They can also modify profiles for existing users. To rephrase, in the User View UI, administrative users can configure managed user data that conforms to the core schema of OpenIDM, including default policies set up in the `policy.json` configuration file. For more information, see Chapter 9, "Using Policies to Validate Data" in the *Integrator's Guide*.

In addition, administrative users can configure and manage workflows in the User View UI. For more information, see Section 4.5, "Managing Workflows From the User View UI" in the *Integrator's Guide*.

In essence, the User View UI supports day-to-day administrative tasks.

In contrast, the Admin UI allows an administrator to define the overall OpenIDM system configuration. Administrators would access the Admin UI to learn OpenIDM, during initial system setup, and when they identify new requirements.

Unlike the User View UI, the Admin UI allows you to configure connections to external data stores, as well as the way OpenIDM reconciles information between internal and external data stores.

When OpenIDM is running on the localhost system, you can access these UIs at <https://localhost:8443/openidmui> and <https://localhost:8443/admin>, respectively.

Chapter 2

First OpenIDM Sample

This chapter provides an overview of the first sample and how it is configured. To see a listing and an overview of the rest of the samples provided, see the README found in [openidm/samples](#) and in the chapter Chapter 3, "*More OpenIDM Samples*".

2.1. Before You Begin

Install OpenIDM as described in Chapter 1, "*Installing OpenIDM Services*".

OpenIDM comes with an internal noSQL database, OrientDB, for use as the internal repository out of the box. This makes it easy to get started with OpenIDM. OrientDB is not yet supported for production use, however, so use a supported JDBC database when moving to production.

If you want to query the internal noSQL database, you can download OrientDB (version 1.7.10) from <http://orientdb.com/download/>. You will find the shell console in the `bin` directory. Start OrientDB console using either `console.sh` or `console.bat`, and then connect to the running OpenIDM with the `connect` command.

```
$ cd /path/to/orientdb-community-1.7.10/bin
$ ./console.sh
OrientDB console v.1.7.10 (build @BUILD@) www.orienttechnologies.com
Type 'help' to display all the commands supported.

Installing extensions for GREMLIN language v.2.5.0-SNAPSHOT

orientdb> connect remote:localhost/openidm admin admin
Connecting to database [remote:localhost/openidm] with user 'admin'...OK

orientdb>
```

When you have connected to the database, you might find the following commands useful.

info

Shows classes and records

select * from managed_user

Shows all users in the OpenIDM repository

select * from audit_activity

Shows all activity audit records

This table is created when there is some activity.

select * from audit_recon

Shows all reconciliation audit records

This table is created when you run reconciliation.

You can also use OrientDB Studio to query the default OrientDB repository. After you have installed and started OpenIDM, point your browser to <http://localhost:2480/>. The default database is `openidm` and the default user and password are `admin` and `admin`. Click Connect to connect to the repository.

To change the default password, use the following POST request on the `repo` endpoint:

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/repo?_action=updateDbCredentials&user=admin&password=newPassword"
```

You must restart OpenIDM for the change to take effect.

This command updates both the repository and the repository configuration file.

2.2. About the Sample

OpenIDM connects data objects held in external resources by mapping one object to another. To connect to external resources, OpenIDM uses OpenICF connectors, configured for use with the external resources.

When objects in one external resource change, OpenIDM determines how the changes affect other objects, and can make the changes as necessary. This sample demonstrates how OpenIDM does this by using *reconciliation*. OpenIDM reconciliation compares the objects in one object set to mapped objects in another object set. Reconciliation can work in write mode, where OpenIDM writes changes to affected objects, or in report mode, where OpenIDM reports on what changes would be written without making the changes. For a complete explanation of reconciliation and synchronization, see the section on Section 12.1, "Types of Synchronization" in the *Integrator's Guide*.

This sample connects to an XML file that holds sample user data. The XML file is configured as the authoritative source. In this sample, users are created in the local repository to show you how you can manage local users through the REST APIs as well as through the OpenIDM UI.

You can also use OpenIDM without storing managed objects for users in the local repository, instead reconciling and synchronizing objects directly through connectors to external resources.

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

Sample Configuration Files

You can find configuration files for the sample under the `openidm/samples/sample1/conf` directory. As you review the sample, keep the following in mind.

1. Start OpenIDM with the configuration associated with Sample 1.

```
$ ./startup.sh -p samples/sample1
```

For more information, see Section 2.3, "Install the Sample".

2. OpenIDM regularly scans for any scheduler configuration files in the `conf` directory.
3. OpenIDM's reconciliation service reads the mappings and actions for the source and target users from `conf/sync.json`.
4. When you initiate a reconciliation, OpenIDM queries all users in the source, and then creates, deletes, or modifies users in the local OpenIDM repository as mapped in `conf/sync.json`.
5. OpenIDM writes all operations to the audit logs in both the internal database and also the flat files in the `openidm/audit` directory.
6. The default Sample 1 version of the `conf/authentication.json` file includes three authentication modules: `MANAGED_USER`, `INTERNAL_USER`, and `CLIENT_CERT`. For more information, see the Integrator's Guide section on Section 15.4, "Using Delegated Authentication" in the *Integrator's Guide*.

When you start OpenIDM with the `-p` project variable (`./startup.sh -p samples/sample1`), the `&{launcher.project.location}` is set to a value of `samples/sample1`. The configuration files use this, as shown in the following sections.

The following configuration files play important roles in this sample.

`samples/sample1/conf/provisioner.openicf-xml.json`

This connector configuration file serves as the XML file resource. It is a copy of the file of the same name found in the `samples/provisioners` directory.

In this sample, the connector instance acts as the authoritative source for users. In the configuration file you can see that the `xmlFilePath` is set to `&{launcher.project.location}/data/xmlConnectorData.xml`.

The `&{launcher.project.location}`, in this case, is `sample/sample1`.

For details on the OpenICF connector configuration files see Chapter 11, "Connecting to External Resources" in the *Integrator's Guide*.

`samples/sample1/conf/schedule-reconcile_systemXmlAccounts_managedUser.json`

The sample schedule configuration file defines a reconciliation job that, if enabled by setting `"enabled" : true`, starts a reconciliation each minute for the mapping named `systemXmlAccounts_managedUser`. The mapping is defined in the configuration file, `conf/sync.json`.

```
{
  "enabled" : false,
  "type": "cron",
  "schedule": "30 0/1 * * * ?",
  "persisted" : true,
  "misfirePolicy" : "fireAndProceed",
  "invokeService": "sync",
  "invokeContext": {
    "action": "reconcile",
    "mapping": "systemXmlfileAccounts_managedUser"
  }
}
```

For information about the schedule configuration see Chapter 13, "Scheduling Tasks and Events" in the *Integrator's Guide*.

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

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/recon?
_action=recon&mapping=systemXmlfileAccounts_managedUser&waitForCompletion=true"
```

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

Note

If you want to set up a self-signed certificate, see Section 16.2.2, "Restrict REST Access to the HTTPS Port" in the *Integrator's Guide*.

Alternatively, just substitute a `-k` or `--insecure` for `--cacert self-signed.crt` in the REST calls (the `curl` commands) described in the OpenIDM documentation.

`samples/sample1/conf/sync.json`

This sample configuration file defines the configuration for reconciliation and synchronization. The `systemXmlAccounts_managedUser` is the mapping for the reconciliation. This entry in `conf/sync.json` defines the synchronization mappings between the XML file connector (source) and the local repository (target).

```
{
  "mappings": [
    {
      "name": "systemXmlfileAccounts_managedUser",
      "source": "system/xmlfile/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" : "source.split(',')"
            },
            "target" : "roles"
        }
    ],
    "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 local OpenIDM repository. Paths that start with `system`, such as `system/xmlfile/account`, refer to connector objects, in this case the XML file connector.

For more information about synchronization, reconciliation, and `sync.json`, see Chapter 12, "Configuring Synchronization" in the *Integrator's Guide*.

For additional examples related to scripting, see the Appendix F, "Scripting Reference" in the *Integrator's Guide*.

2.3. Install the Sample

Now that OpenDJ is configured, prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM". You can then start OpenIDM with the configuration for Sample 1.

```
$ cd /path/to/openidm  
$ ./startup.sh -p samples/sample1
```

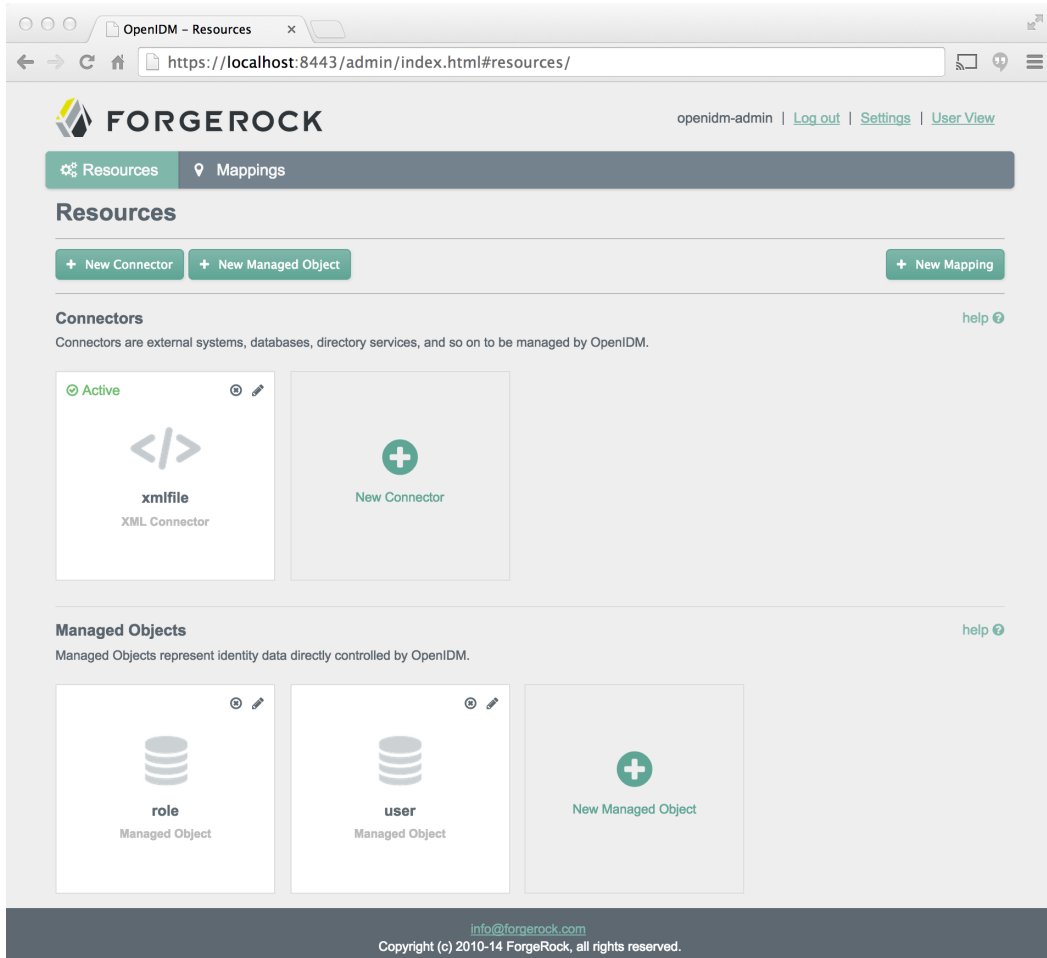
2.4. Review the Sample in the Administrative User Interface

OpenIDM includes a web-based Administrative User Interface, known as the "Admin UI". For details, see Section 4.1, "Configuring OpenIDM from the Admin UI" in the *Integrator's Guide*.

After starting OpenIDM, you can 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`). After logging in, OpenIDM asks you to change the credentials for the

administrative account, which is a good idea. However, you can bypass this request by closing that window.

You should now see the Resources screen, with the connectors and managed objects associated with that configuration.



2.5. Running Reconciliation

Start OpenIDM with the configuration for sample 1.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/sample1
```

Reconcile the objects in the resources, either by setting `"enabled" : true` in the schedule configuration file (`conf/schedule-reconcile_systemXmlAccounts_managedUser.json`) and then waiting until the scheduled reconciliation happens, or by using the REST interface, as shown in the following example.

```
$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --header "Content-Type: application/json" \
  --request POST \
  "https://localhost:8443/openidm/recon?
  _action=recon&mapping=systemXmlfileAccounts_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"
}
```

Alternatively, you can run the same reconciliation in the Admin UI. To do so, select Mappings. For Sample 1, you should see one mapping, `systemXmlfileAccounts_managedUser`. Select Edit to access configuration options associated with reconciliation. To run the reconciliation, click Reconcile Now.

systemXmlfileAccounts_managedUser

Source: **system/xmlfile/account** (xmlfile) → Target: **managed/user** (managed)

Completed: Last Reconciled December 01, 2014 15:07 Reconcile Now help

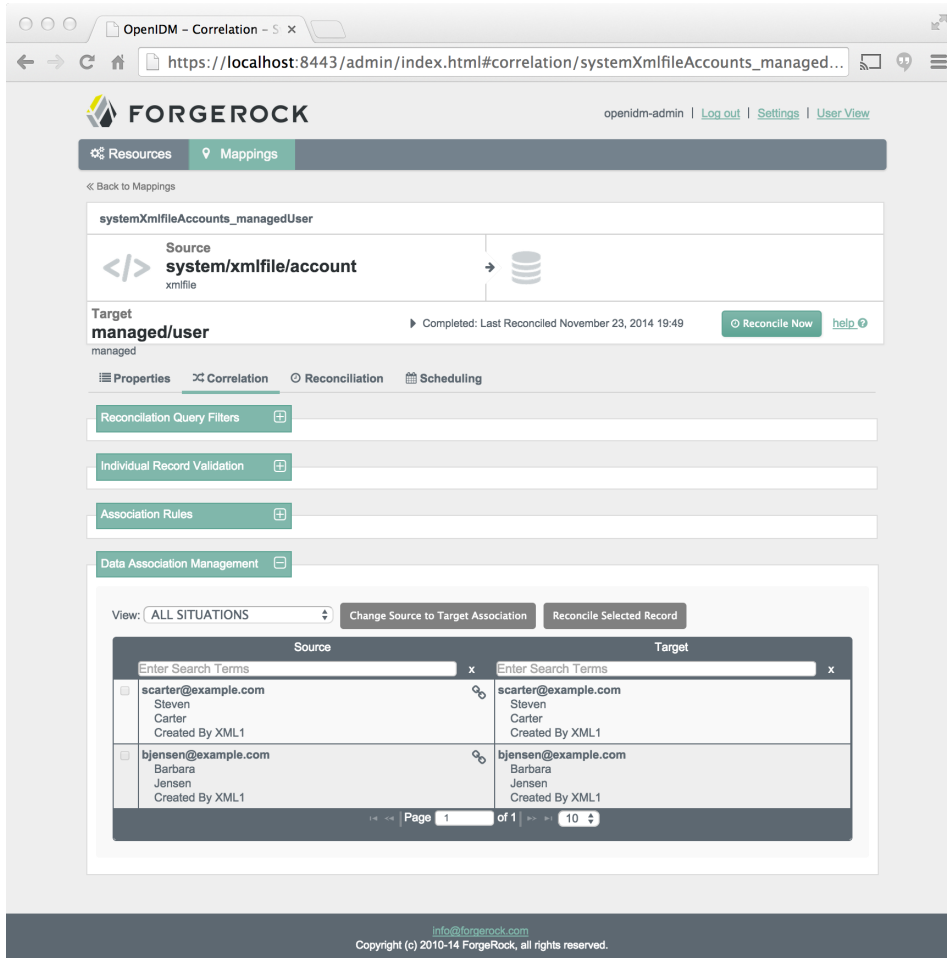
Number of representative properties: 4 Sample Source:

Source	Target	Default	Transform Script	Sample
email	mail			
firstname	givenName			
lastname	sn			
description	description			
_id	_id			
name	userName			
password	password			
mobileTelephoneNumber	telephoneNumber			
roles	roles		source.split(',')	

Clear Changes Save Properties

2.6. Viewing Users and Logs

After reconciliation, you can use the Admin UI to display user records in both the source and target resources. Navigate to the URL where OpenIDM is installed. If it is local, navigate to <https://localhost:8443/admin>. Select mappings, you should see the result of the reconciliation.



Alternatively, you can use the REST interface to display all users in the local repository. Use a REST client to perform an HTTP GET on the following URL: https://localhost:8443/openidm/managed/user?_queryId=query-all-ids with the headers "X-OpenIDM-Username: openidm-admin" and "X-OpenIDM-Password: openidm-admin".

OpenIDM returns JSON data. Depending on the browser, you can use a REST client to display the JSON or download it as a file. Alternatively, you can use the following `curl` command to get the JSON response.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/managed/user?_queryId=query-all-ids"

{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 2,
  "result": [
    {
      "_rev": "0",
      "_id": "scarter"
    },
    {
      "_rev": "0",
      "_id": "bjensen"
    }
  ]
}
```

You can set up arbitrary queries. For more information about using query expressions in a REST call, see Section 7.3, "Defining and Calling Queries" in the *Integrator's Guide*.

If you created user `joe`, as described in Section 1.3, "To Get Started With the OpenIDM REST Interface", you should see the ID for that user somewhere in this list. If you did not include `"_id:joe"` in the command to create user `joe`, you would see a system-generated UUID in the list instead of the specified ID.

Now try a RESTful GET of user `bjensen` by appending the user ID to the managed user URL (<https://localhost:8443/openidm/managed/user/>).

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/managed/user/bjensen"

{
  "mail" : "bjensen@example.com",
  "sn" : "Jensen",
  "passwordAttempts" : "0",
  "lastPasswordAttempt" : "Tue Apr 15 2014 20:58:46 GMT-0800 (GMT-08:00)",
  "address2" : "",
  "givenName" : "Barbara",
  "effectiveRoles" : [ "openidm-authorized" ],
  "country" : "",
  "city" : "",
  "lastPasswordSet" : "",
  "postalCode" : "",
  "_id" : "bjensen",
  "_rev" : "1",
  "description" : "Created By XML1",
  "accountStatus" : "active",
  "telephoneNumber" : "1234567",
  "roles" : [ "openidm-authorized" ],
  "postalAddress" : "",
  "stateProvince" : "",
  "userName" : "bjensen@example.com"
}
```

The complete user record is returned. If you need this level of information for all users, substitute `query-all` for `query-all-ids`.

So with some of the query expressions described in the Section 7.3, "Defining and Calling Queries" in the *Integrator's Guide*, you can filter the output.

As defined in the mapping file `conf/sync.json`, the `sn` and `mail` parameters correspond to surname (sn) and email address, respectively.

For example, the following RESTful GET filters output by surname (sn):

```
$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --request GET \
  "https://localhost:8443/openidm/managed/user?_queryFilter=sn%20%pr&_fields=sn"

{
  "result" : [ {
    "sn" : "Carter",
  }, {
    "sn" : "Jensen"
  } ]
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : -1
}
```

Now that you have a listing of desired users, you can add more fields:

```
$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --request GET \
  "https://localhost:8443/openidm/managed/user?_queryFilter=sn%20%pr&_fields=sn,mail,description"

{
  "result" : [ {
    "sn" : "Carter",
    "mail" : "scarter@example.com",
    "description" : "Created by XML1",
  }, {
    "sn" : "Jensen"
    "mail" : "bjensen@example.com",
    "description" : "Created by XML1",
  } ]
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : -1
}
```

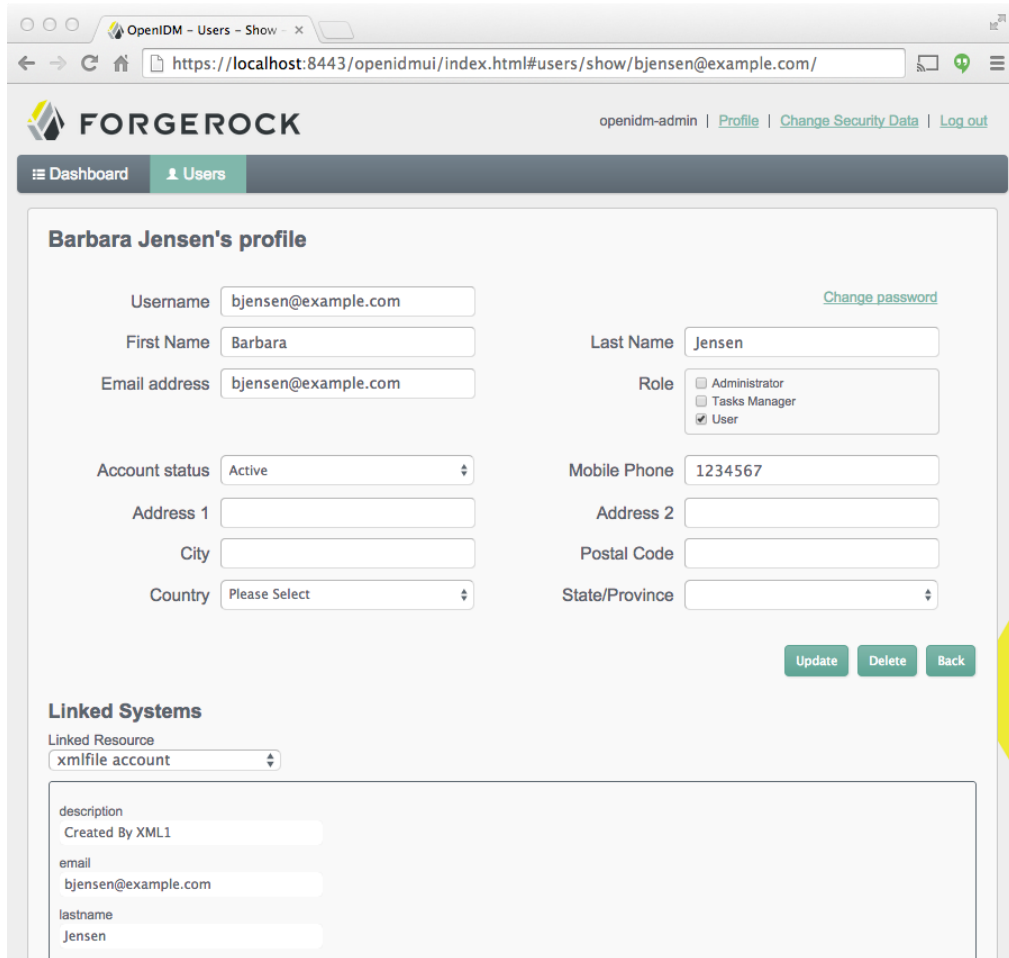
This information is also available in the CSV format audit logs located in the `openidm/audit` directory.

```
$ ls /path/to/openidm/audit/

access.csv activity.csv recon.csv
```

For more information on what you can find in each of these files, see Section 18.1, "Audit Log Types" in the *Integrator's Guide*.

You can get a similar level of information for each user. For example, after reconciliation, you can return to Section 2.6, "Viewing Users and Logs", select a user, and review information from the reconciled linked resource.



2.7. Adding Users in a Resource

Add a user to the source connector XML data file to see reconciliation in action. During the next reconciliation, OpenIDM finds the new user in the source connector, and creates the user in the local repository. To add the user, copy the following XML into `openidm/samples/sample1/data/xmlConnectorData.xml`.

```
<ri: __ACCOUNT__ >
  <icf: __UID__ >tmorris</icf: __UID__ >
  <icf: __NAME__ >tmorris@example.com</icf: __NAME__ >
  <ri:password>TestPassw0rd#</ri:password>
  <ri:firstname>Toni</ri:firstname>
  <ri:lastname>Morris</ri:lastname>
  <ri:email>tmorris@example.com</ri:email>
  <ri:mobileTelephoneNumber>1234567</ri:mobileTelephoneNumber>
  <ri:roles>openidm-authorized</ri:roles>
  <icf: __DESCRIPTION__ >Created By XML1</icf: __DESCRIPTION__ >
</ri: __ACCOUNT__ >
```

Run reconciliation again, as described in Section 2.5, "Running Reconciliation". After reconciliation has run, query the local repository to see the new user appear in the list of all users under https://localhost:8443/openidm/managed/user?_queryId=query-all-ids.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/managed/user?_queryId=query-all-ids"

{
  "result": [ {
    "_id": "bjensen",
    "_rev": "0"
  }, {
    "_id": "scarter",
    "_rev": "0"
  }, {
    "_id": "tmorris",
    "_rev": "0"
  } ],
  "resultCount": 3,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
```

Also look at the reconciliation audit log, [openidm/audit/recon.csv](#) to see what took place during reconciliation. This formatted excerpt from the log covers the two reconciliation runs done in this sample.

```
"_id", "action", ..., "reconId", "situation", "sourceObjectId", "targetObjectId", "timestamp";
"7e...", "CREATE", ..., "486...", "ABSENT", "system/xmlfile/account/bjensen", "managed/user/bjensen", ...;
"1a...", "CREATE", ..., "486...", "ABSENT", "system/xmlfile/account/scarter", "managed/user/scarter", ...;
"33...", "UPDATE", ..., "aa9...", "CONFIRMED", "system/xmlfile/account/bjensen", "managed/user/bjensen", ...;
"1d...", "UPDATE", ..., "aa9...", "CONFIRMED", "system/xmlfile/account/scarter", "managed/user/scarter", ...;
"0e...", "CREATE", ..., "aa9...", "ABSENT", "system/xmlfile/account/tmorris", "managed/user/tmorris", ...;
```

The relevant audit log fields in this example are: action, situation, [sourceObjectId](#), and [targetObjectId](#). For each object in the source, reconciliation leads to an action on the target.

In the first reconciliation run (the abbreviated `reconID` is shown as `486...`), the source object does not exist in the target, resulting in an ABSENT situation and an action to CREATE the object in the target. The object created earlier in the target does not exist in the source, and so is IGNORED.

In the second reconciliation run (the abbreviated `reconID` is shown as `aa9...`), after you added a user to the source XML, OpenIDM performs an UPDATE on the user objects `bjensen` and `scarter` that already exist in the target, in this case changing the internal ID. OpenIDM performs a CREATE on the target for the new user (`tmorris`).

You configure the action that OpenIDM takes based on an object's situation in the configuration file, `conf/sync.json`. For the list of all possible situations and actions, see Chapter 12, "Configuring Synchronization" in the *Integrator's Guide*.

For details on auditing, see Chapter 18, "Using Audit Logs" in the *Integrator's Guide*.

2.8. Adding Users Over REST

You can add users to the local repository over the REST interface. The following example adds a user named James Berg.

Create `james` (UNIX).

```
$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --header "Content-Type: application/json" \
  --request POST \
  --data '{
    "_id": "jberg",
    "userName": "jberg",
    "sn": "Berg",
    "givenName": "James",
    "mail": "jberg@example.com",
    "telephoneNumber": "5556787",
    "description": "Created by OpenIDM REST.",
    "password": "MyPassw0rd"
  }' \
  "https://localhost:8443/openidm/managed/user?_action=create"

{
  "mail" : "jberg@example.com",
  "sn" : "Berg",
  "passwordAttempts" : "0",
  "lastPasswordAttempt" : "Tue Apr 15 2014 21:05:12 GMT-0800 (GMT-08:00)",
  "address2" : "",
  "givenName" : "James",
  "effectiveRoles" : [ "openidm-authorized" ],
  "password" : {
    "$crypto" : {
      "type" : "x-simple-encryption",
      "value" : {
```



```

        "data" : "QYRcIS9FbksBEwyd4dNEpg==",
        "cipher" : "AES/CBC/PKCS5Padding",
        "iv" : "R5Kjs6jZZtqCockFCS6BfA==",
        "key" : "openidm-sym-default"
    }
}
},
"country" : "",
"city" : "",
"_rev" : "1",
"lastPasswordSet" : "",
"postalCode" : "",
"_id" : "jberg",
"description" : "Created by OpenIDM REST.",
"accountStatus" : "active",
"telephoneNumber" : "5556787",
"roles" : [ "openidm-authorized" ],
"effectiveAssignments" : { },
"postalAddress" : "",
"stateProvince" : "",
"userName" : "jberg"
}

```

Create **james** (Windows).

```

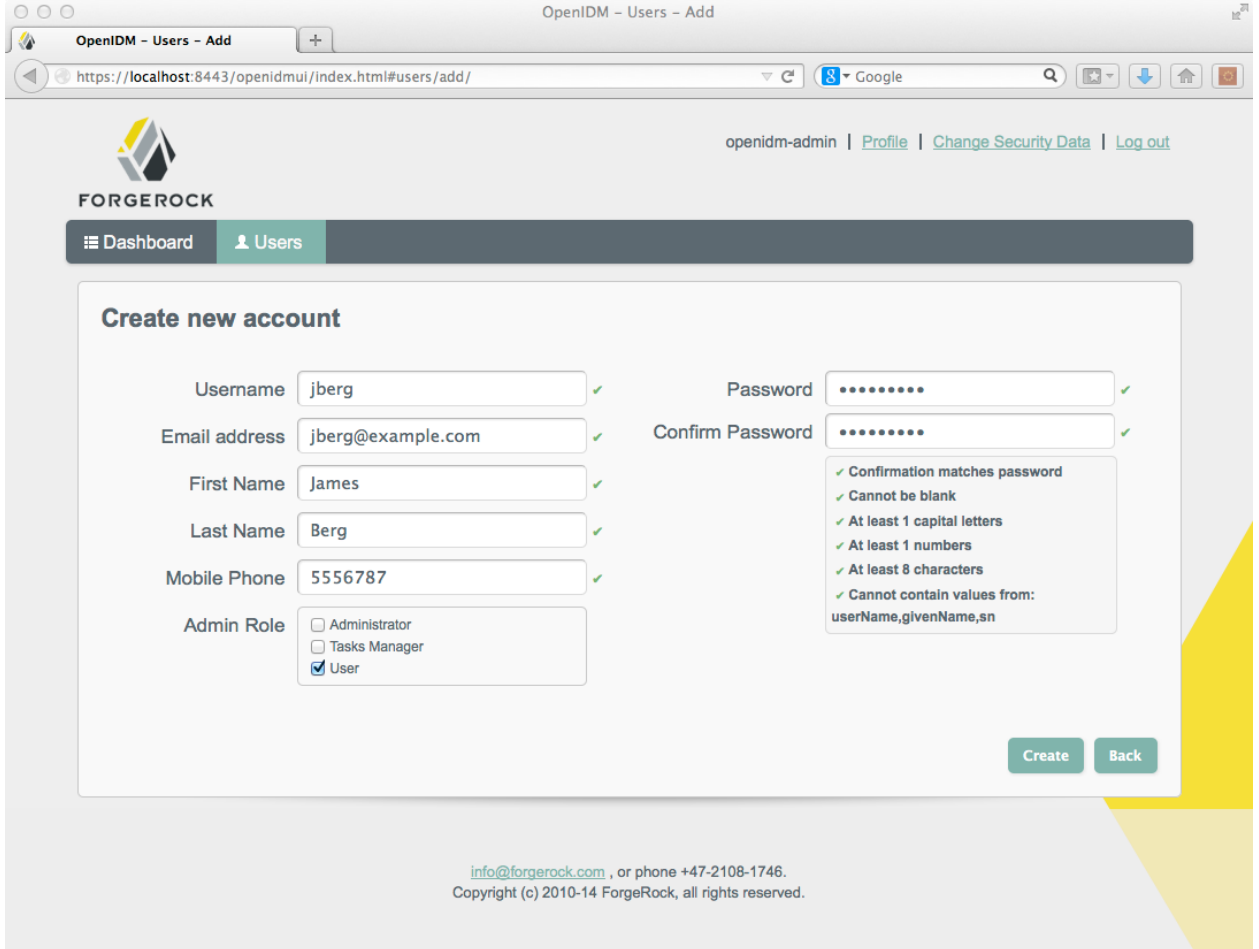
C:\> curl
--cacert self-signed.crt
--header "X-OpenIDM-Username: openidm-admin"
--header "X-OpenIDM-Password: openidm-admin"
--header "Content-Type: application/json"
--request POST
--data "{
\"_id\": \"jberg\",
\"userName\": \"jberg\",
\"sn\": \"Berg\",
\"givenName\": \"James\",
\"email\": \"jberg@example.com\",
\"telephoneNumber\": \"5556787\",
\"description\": \"Created by OpenIDM REST.\",
\"password\": \"MyPassw0rd\"
}"
"https://localhost:8443/openidm/managed/user?_action=create"

```

The output is essentially identical to that for UNIX.

OpenIDM creates the new user in the repository. If you configure a mapping to apply changes from the local repository to the XML file connector as a target, OpenIDM then updates the XML file to add the new user.

You can also add users through the UI, which uses the OpenIDM REST API. When you have logged into the UI as the OpenIDM administrator, the process is straightforward, as shown in the following figure.



The screenshot shows a web browser window with the URL `https://localhost:8443/openidm/ui/index.html#users/add/`. The page title is "OpenIDM - Users - Add". The interface includes a navigation bar with "Dashboard" and "Users" (selected). The main content area is titled "Create new account" and contains a form with the following fields and values:

- Username: `jberg` ✓
- Email address: `jberg@example.com` ✓
- First Name: `James` ✓
- Last Name: `Berg` ✓
- Mobile Phone: `5556787` ✓
- Admin Role: Administrator, Tasks Manager, User
- Password: `.....` ✓
- Confirm Password: `.....` ✓

Validation messages for the password field:

- ✓ Confirmation matches password
- ✓ Cannot be blank
- ✓ At least 1 capital letters
- ✓ At least 1 numbers
- ✓ At least 8 characters
- ✓ Cannot contain values from: `userName,givenName,sn`

Buttons: "Create" and "Back".

Footer: info@forgerock.com, or phone +47-2108-1746.
Copyright (c) 2010-14 ForgeRock, all rights reserved.

Chapter 3

More OpenIDM Samples

The current distribution of OpenIDM comes with a variety of samples in `openidm/samples/`. Sample 1 is described in Chapter 2, "*First OpenIDM Sample*". This chapter describes the remaining OpenIDM samples.

3.1. Before You Begin

Install OpenIDM, as described in Chapter 1, "*Installing OpenIDM Services*".

OpenIDM comes with an internal noSQL database, OrientDB, for use as the internal repository out of the box. This makes it easy to get started with OpenIDM. OrientDB is not yet supported for production use, however, so use a supported JDBC database when moving to production.

3.1.1. Installing the Samples

Each sample directory in `openidm/samples/` contains a number of subdirectories, such as `conf/` and `script/`. To start OpenIDM 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, but not all samples require additional software, such as an external LDAP server or database.

When you move from one sample to the next, bear in mind that you are changing the OpenIDM configuration. For information on how configuration changes work, see Section 6.2, "*Changing the Default Configuration*" in the *Integrator's Guide*.

The command-line examples in this chapter (and throughout the OpenIDM documentation) assume a UNIX shell. If you are running these samples on Windows, adjust the command-line examples accordingly. For an indication of what the corresponding Windows command would look like, see the examples in the Chapter 2, "*First OpenIDM Sample*".

3.1.2. Preparing OpenIDM

Install an instance of OpenIDM specifically to try the samples. That way you can experiment as much as you like, and discard the result if you are not satisfied.

If you are using the same instance of OpenIDM for multiple samples, it is helpful to clear out the repository created for an earlier sample. To do so, shut down OpenIDM and delete the `openidm/db/openidm` directory.

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

OpenIDM should now be ready to start with a new sample. For a number of the following samples, users are created either at the UI or with a commons REST call. Once added, and when reconciliation is complete, such users should be able to log into the UI.

3.2. Sample 1 - XML File

Sample 1 is described in Chapter 2, "*First OpenIDM Sample*".

3.3. Sample 2 - LDAP One Way

Sample 2 resembles the first sample, but in sample 2 OpenIDM is connected to a local LDAP server. The sample has been tested with OpenDJ, but should work with any LDAPv3 compliant server.

Sample 2 demonstrates how OpenIDM can pick up new or changed objects from an external resource. The sample contains only one mapping, from the external LDAP server resource to the OpenIDM repository. The sample therefore does not push any changes made to OpenIDM managed user objects out to the LDAP server.

3.3.1. LDAP Server Configuration

Sample 2 expects the following configuration for the external LDAP server:

- The LDAP server runs on the local host.
- The LDAP server listens on port 1389.
- A user with DN `cn=Directory Manager` and password `password` has read access to the LDAP server.
- Directory data for that server is stored under base DN `dc=example,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`

An example user object follows.

```
dn: uid=jdoe,ou=People,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
givenName: John
uid: jdoe
cn: John Doe
telephoneNumber: 1-415-523-0772
sn: Doe
mail: jdoe@example.com
description: Created by OpenIDM
userPassword: password
```

A sample LDIF file, with two users, is provided in [openidm/samples/sample2/data/Example.ldif](#).

The following steps provide setup instructions for an OpenDJ server. Adjust these instructions if you are using an alternative LDAP server.

1. Download and extract the OpenDJ zip archive from the [OpenDJ Project](#) page.
2. Install OpenDJ using the command-line setup. During the install, import the LDIF data required for this sample:

```
$ cd /path/to/opensj
$ ./setup --cli
\
--hostname localhost
\
--ldapPort 1389
\
--rootUserDN "cn=Directory Manager"
\
--rootUserPassword password
\
--adminConnectorPort 4444
\
--baseDN dc=com
\
--ldifFile /path/to/openidm/samples/sample2/data/Example.ldif
\
--acceptLicense
\
--no-prompt
...
Configuring Directory Server ..... Done.
Importing LDIF file /path/to/openidm/samples/sample2/data/Example.ldif ..... Done.
Starting Directory Server ..... Done.
.
...
```

3.3.2. Install the Sample

Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM", then start OpenIDM with the configuration for sample 2.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/sample2
```

3.3.3. Reconcile the Repository

The mapping configuration file ([sync.json](#)) for this sample includes the mapping `systemLdapAccounts_managedUser`, which synchronize users from the source LDAP server with the target OpenIDM repository.

You can run this part of the sample by using the `curl` command-line utility, or by using the OpenIDM Administration UI. This section provides instructions for both methods.

Procedure 3.1. To Run the Sample Using the Command Line

1. Reconcile the repository by running the following command:

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/recon?
action=recon&mapping=systemLdapAccounts_managedUser&waitForCompletion=true"
{
  "state": "SUCCESS",
  "_id": "f3c618aa-cc3b-49ed-9a3a-00b012db2513"
}
```

The reconciliation operation creates the two users from the LDAP server in the OpenIDM repository, assigning the new objects random unique IDs.

- To retrieve the users from the repository, query their IDs as follows:

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/managed/user?_queryId=query-all-ids"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 2,
  "result": [
    {
      "_rev": "0",
      "_id": "0a5546d6-149b-4f8b-b3be-4afa8a267d45"
    },
    {
      "_rev": "0",
      "_id": "840394b9-ced0-4f13-a5ad-6a0d825f0705"
    }
  ]
}
```

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

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/managed/user/0a5546d6-149b-4f8b-b3be-4afa8a267d45"
{
  "displayName": "Babara Jensen",
  "stateProvince": "",
  "userName": "bjensen",
  "postalAddress": "",
  "effectiveAssignments": {},
  "roles": [
    "openidm-authorized"
  ]
}
```

```

    ],
    "country": "",
    "effectiveRoles": [
      "openidm-authorized"
    ],
    "givenName": "Barbara",
    "address2": "",
    "lastPasswordAttempt": "Mon Nov 24 2014 21:17:19 GMT+0200 (SAST)",
    "passwordAttempts": "0",
    "sn": "Jensen",
    "mail": "bjensen@example.com",
    "city": "",
    "lastPasswordSet": "",
    "postalCode": "",
    "_id": "0a5546d6-149b-4f8b-b3be-4afa8a267d45",
    "_rev": "1",
    "accountStatus": "active",
    "description": "Created for OpenIDM",
    "telephoneNumber": "1-360-229-7105"
  }

```

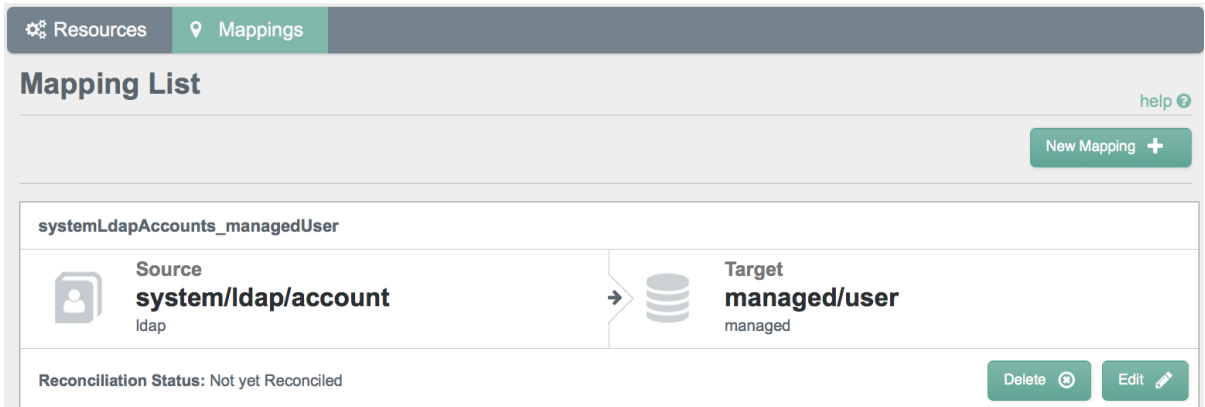
Procedure 3.2. To Run the Sample Using the Admin UI

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`.

The first time you log into the Admin UI, you are prompted to change your password. If you do not want to change your password at this time, simply click X to close this window, and continue with the sample.

2. Select the Mappings tab.

This tab shows one configured mapping, from the `ldap` server to the OpenIDM repository (`managed/user`).



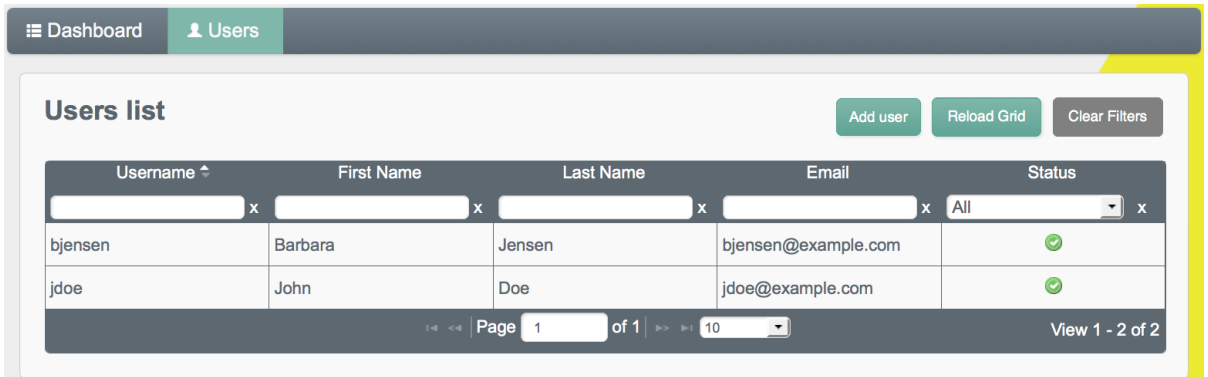
3. Click anywhere on the mapping and click Reconcile Now.

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

- Retrieve the users in the repository by clicking the User View link at the top right of the Admin UI.

This link opens the User View UI. If you did not change your password in the first step, you are prompted to change your password again. You can bypass this by simply clicking X to close the password prompt window.

- Select the Users tab.



The two users from the LDAP server have been reconciled to the OpenIDM repository.

- To retrieve the details of a specific user, click that username on the Users tab.

The following image shows the details of user **bjensen**.

The screenshot shows the 'Users' section of the OpenIDM dashboard. The profile for 'Barbara Jensen' is displayed with various fields for editing. The fields are organized into two columns. The left column includes Username (bjensen), First Name (Barbara), Email address (bjensen@example.com), Account status (Active), Address 1, City, and Country (Please Select). The right column includes Last Name (Jensen), Role (User selected), Mobile Phone (1-360-229-7105), Address 2, Postal Code, and State/Province. A 'Change password' link is located next to the Username field. At the bottom right, there are three buttons: 'Update', 'Delete', and 'Back'.

3.4. Sample 2b - LDAP Two Way

Like sample 2, sample 2b connects to an external LDAP server. However, sample 2b has two mappings configured, one from the LDAP server to the OpenIDM repository, and the other from the OpenIDM repository to the LDAP server.

3.4.1. External LDAP Configuration

Configure the LDAP server as for sample 2, Section 3.3.1, "LDAP Server Configuration". The LDAP user must have write access to create users from OpenIDM on the LDAP server. When you configure the LDAP server, import the appropriate LDIF file ([openidm/samples/sample2b/data/Example.ldif](#)).

3.4.2. Install the Sample

Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM", then start OpenIDM with the configuration for sample 2b.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/sample2b
```

3.4.3. Run the Sample

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 OpenIDM repository, and `managedUser_systemLdapAccounts`, which synchronizes changes from the OpenIDM repository to the LDAP server.

You can run this part of the sample by using the `curl` command-line utility, or by using the OpenIDM Administration UI. This section provides instructions for both methods.

Procedure 3.3. To Run the Sample Using the Command Line

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

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/recon?
action=recon&mapping=systemLdapAccounts_managedUser&waitForCompletion=true"
{
  "state": "SUCCESS",
  "_id": "027e25e3-7a33-4858-9080-161c2b40a6bf"
}
```

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

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

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/managed/user?_queryId=query-all-ids"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 2,
  "result": [
    {
      "_rev": "0",
      "_id": "2cb34789-ee67-4f50-88f2-022460c37253"
    },
    {
      "_rev": "0",
      "_id": "625942ac-3cb8-47aa-9de1-217d24967b3b"
    }
  ]
}
```

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

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/managed/user/625942ac-3cb8-47aa-9de1-217d24967b3b"
{
  "displayName": "Babara Jensen",
  "stateProvince": "",
  "userName": "bjensen",
  "postalAddress": "",
  "effectiveAssignments": {},
  "roles": [
    "openidm-authorized"
  ],
  "country": "",
  "effectiveRoles": [
    "openidm-authorized"
  ],
  "givenName": "Barbara",
  "address2": "",
  "lastPasswordAttempt": "Tue Nov 25 2014 12:29:15 GMT+0200 (SAST)",
  "passwordAttempts": "0",
  "sn": "Jensen",
  "mail": "bjensen@example.com",
  "city": "",
  "lastPasswordSet": "",
  "postalCode": "",
  "_id": "625942ac-3cb8-47aa-9de1-217d24967b3b",
  "_rev": "1",
  "description": "Created for OpenIDM",
  "accountStatus": "active",
  "telephoneNumber": "1-360-229-7105"
}
```

4. Test the second mapping by creating a user in the OpenIDM repository.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--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"}' \
"https://localhost:8443/openidm/managed/user?_action=create"
{
  "displayName": "fdoe",
  "stateProvince": "",
  "userName": "fdoe",
  "postalAddress": "",

```

```
"effectiveAssignments": {},
"roles": [
  "openidm-authorized"
],
"country": "",
"effectiveRoles": [
  "openidm-authorized"
],
"givenName": "Felicitas",
"address2": "",
"lastPasswordAttempt": "Tue Nov 25 2014 12:49:06 GMT+0200 (SAST)",
"passwordAttempts": "0",
"sn": "Doe",
"mail": "fdoe@example.com",
"city": "",
"_rev": "1",
"lastPasswordSet": "",
"postalCode": "",
"_id": "c462fac4-528a-45c5-ba76-d2744952d418",
"description": "Felicitas Doe",
"accountStatus": "active",
"telephoneNumber": "555-1234"
}
```

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 Section 12.3.2, "Synchronization Mappings File" in the *Integrator's Guide*.

Test that the implicit reconciliation has been successful by querying the users in the LDAP directory over REST, as follows:

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/ldap/account?_queryId=query-all-ids"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 3,
  "result": [
    {
      "_id": "uid=jdoe,ou=People,dc=example,dc=com",
      "dn": "uid=jdoe,ou=People,dc=example,dc=com"
    },
    {
      "_id": "uid=bjensen,ou=People,dc=example,dc=com",
      "dn": "uid=bjensen,ou=People,dc=example,dc=com"
    },
    {
      "_id": "uid=fdoe,ou=People,dc=example,dc=com",
      "dn": "uid=fdoe,ou=People,dc=example,dc=com"
    }
  ]
}
```

Note the new entry for user `fdoe`.

6. Query the complete entry by including `fdoe`'s ID in the URL.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/ldap/account/uid=fdoe,ou=People,dc=example,dc=com"
{
  "_id": "uid=fdoe,ou=People,dc=example,dc=com",
  "dn": "uid=fdoe,ou=People,dc=example,dc=com",
  "mail": "fdoe@example.com",
  "uid": "fdoe",
  "ldapGroups": [],
  "cn": "fdoe",
  "givenName": "Felicitas",
  "telephoneNumber": "555-1234",
  "description": "Felicitas Doe",
  "sn": "Doe"
}
```

Procedure 3.4. To Run the Sample Using 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`.

The first time you log into the Admin UI, you are prompted to change your password. If you do not want to change your password at this time, simply click X to close this window, and continue with the sample.

2. Select the Mappings tab.

This tab shows two configured mappings, one from the `ldap` server to the OpenIDM repository (`managed/user`) and one from the OpenIDM repository to the `ldap` server.

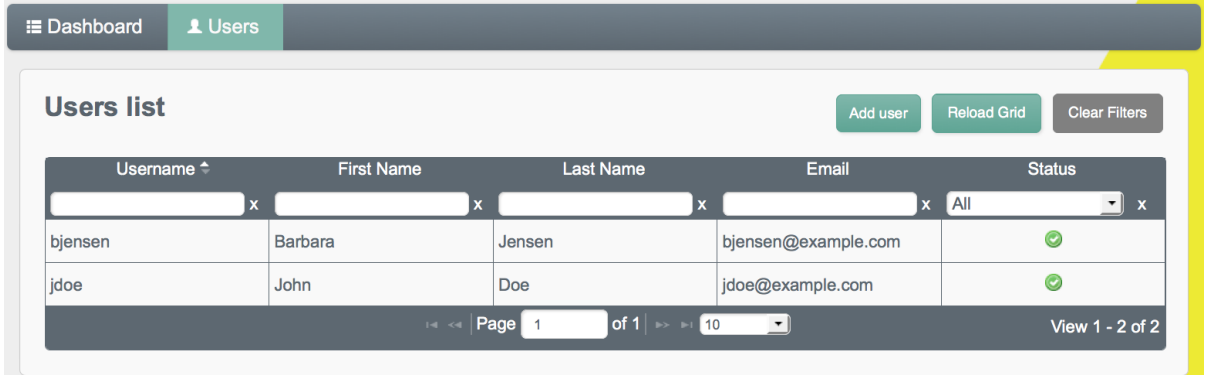
3. Click anywhere on the first mapping and click Reconcile Now.

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

4. Retrieve the users in the repository by clicking the User View link at the top right of the Admin UI.

This link opens the User View UI. If you did not change your password in the first step, you are prompted to change your password again. You can bypass this by simply clicking X to close the password prompt window.

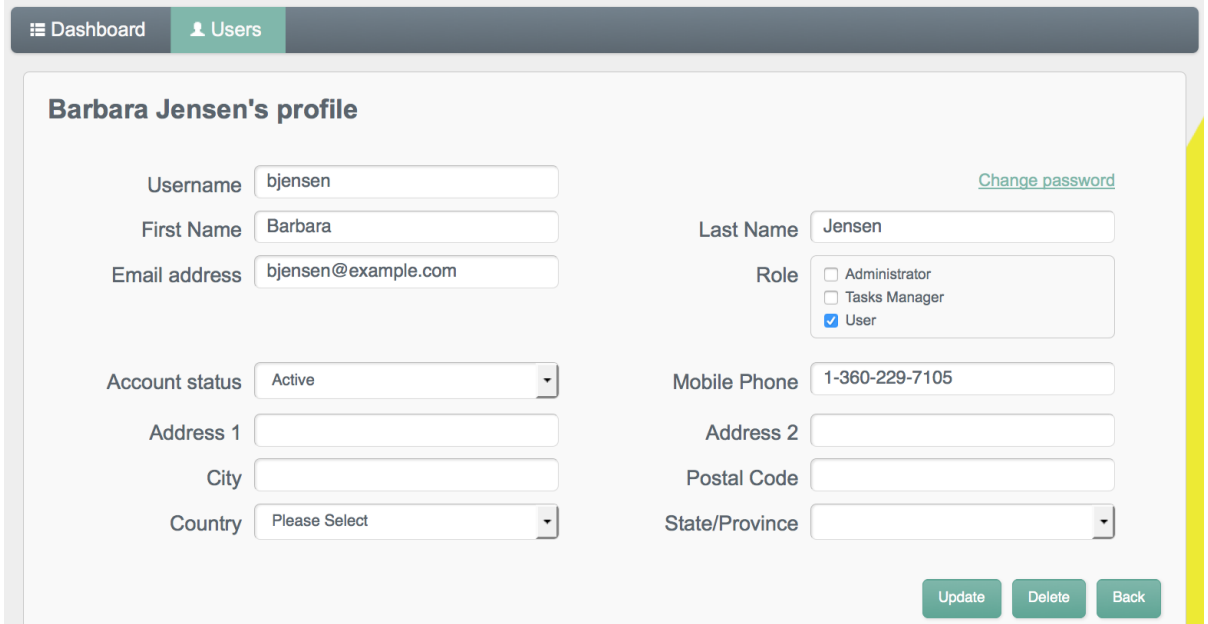
5. Select the Users tab.



The two users from the LDAP server have been reconciled to the OpenIDM repository.

6. To retrieve the details of a specific user, click that username on the Users tab.

The following image shows the details of user **bjensen**.



7. Add a new user in the OpenIDM repository by clicking Add User on the Users tab.

Complete the user details and click Create.

The following image shows a new record being created for user `fdoe`.

Dashboard Users

Create new account

Username

Email address

First Name

Last Name

Mobile Phone

Role

- Administrator
- Tasks Manager
- User

Password

Confirm Password

- ✓ Confirmation matches password
- ✓ Cannot be blank
- ✓ At least 1 capital letters
- ✓ At least 1 numbers
- ✓ At least 8 characters
- ✓ Cannot contain values from: userName, givenName, sn

Create Back

8. 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 Section 12.3.2, "Synchronization Mappings File" in the *Integrator's Guide*.

To test that the implicit reconciliation has been successful, look at `fdoe`'s record in the User View UI. At the bottom of the user profile, the Linked Systems panel indicates the external resource to which this user entry is mapped.

The following image shows that `fdoe` now exists in the linked resource `ldap account`, and shows her entry as it appears on the LDAP server.

Linked Systems

Linked Resource
ldap account

ldapGroups
Collapse

uid
fdoe

cn
Felicitas Doe

telephoneNumber
555-1234

description

dn
uid=fdoe,ou=People,dc=example,dc=com

sn
Doe

givenName
Felicitas

mail
fdoe@example.com

3.5. Sample 2c - Synchronizing LDAP Group Membership

Like sample 2b, sample 2c connects to an external LDAP server and has mappings from the LDAP server to the OpenIDM repository, and from the OpenIDM repository to the LDAP server. However, in sample 2c, LDAP group memberships are synchronized, in addition to user entries.

3.5.1. External LDAP Configuration

Configure the LDAP server as for sample 2, Section 3.3.1, "LDAP Server Configuration". The LDAP user must have write access to create users from OpenIDM on the LDAP server. When you configure the LDAP server, import the appropriate LDIF file ([openidm/samples/sample2c/data/Example.ldif](#)). This file includes the following two LDAP groups:

```
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.

3.5.2. Install the Sample

Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM", then start OpenIDM with the configuration for sample 2c.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/sample2c
```

3.5.3. Run the Sample

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 OpenIDM repository, and `managedUser_systemLdapAccounts`, which synchronizes changes from the OpenIDM repository to the LDAP server.

You can run this part of the sample by using the `curl` command-line utility, or by using the OpenIDM Administration UI. This section provides instructions for both methods.

Procedure 3.5. To Run the Sample Using the Command Line

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

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/recon?
action=recon&mapping=systemLdapAccounts_managedUser&waitForCompletion=true"
{
  "state": "SUCCESS",
  "_id": "8a25d1ca-c0f3-4e0f-82a0-b5c346282782"
}
```

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

- To retrieve the users from the repository, query their IDs as follows:

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/managed/user?_queryId=query-all-ids"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 2,
  "result": [
    {
      "_rev": "0",
      "_id": "40f41f52-44a0-40fe-83ab-a95b92a25805"
    },
    {
      "_rev": "0",
      "_id": "cec9e2d7-cb6c-4622-94a8-2cb43695476f"
    }
  ]
}
```

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

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/managed/user/cec9e2d7-cb6c-4622-94a8-2cb43695476f"
{
  "displayName": "John Doe",
  "userName": "jdoe",
  "stateProvince": "",
  "postalAddress": "",
  "ldapGroups": [
    "cn=openidm,ou=Groups,dc=example,dc=com"
  ]
}
```

```

    ],
    "effectiveAssignments": {},
    "roles": [
      "openidm-authorized"
    ],
    "country": "",
    "effectiveRoles": [
      "openidm-authorized"
    ],
    "givenName": "John",
    "address2": "",
    "lastPasswordAttempt": "Wed Nov 26 2014 14:30:58 GMT+0200 (SAST)",
    "passwordAttempts": "0",
    "sn": "Doe",
    "mail": "jdoe@example.com",
    "city": "",
    "lastPasswordSet": "",
    "postalCode": "",
    "_id": "cec9e2d7-cb6c-4622-94a8-2cb43695476f",
    "_rev": "1",
    "description": "Created for OpenIDM",
    "accountStatus": "active",
    "telephoneNumber": "1-415-599-1100"
  }
}

```

Note that John Doe's user object contains an `ldapGroups` property, the value of which indicates 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 \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --header "Content-Type: application/json" \
  --request POST \
  --data '[
    {
      "operation": "replace",
      "field": "/ldapGroups",
      "value": ["cn=openidm2,ou=Groups,dc=example,dc=com"]
    }
  ]' \
  "https://localhost:8443/openidm/managed/user?_action=patch&_queryId=for-username&uid=jdoe"
{
  "displayName": "John Doe",
  "userName": "jdoe",
  "stateProvince": "",
  "postalAddress": "",
  "effectiveAssignments": {},
  "ldapGroups": [
    "cn=openidm2,ou=Groups,dc=example,dc=com"
  ],
  "roles": [
    "openidm-authorized"
  ]
}

```

```

    ],
    "city": "",
    "effectiveRoles": [
      "openidm-authorized"
    ],
    "givenName": "John",
    "lastPasswordAttempt": "Wed Nov 26 2014 14:30:58 GMT+0200 (SAST)",
    "address2": "",
    "passwordAttempts": "0",
    "sn": "Doe",
    "mail": "jdoe@example.com",
    "country": "",
    "_rev": "2",
    "lastPasswordSet": "",
    "postalCode": "",
    "_id": "cec9e2d7-cb6c-4622-94a8-2cb43695476f",
    "description": "Created for OpenIDM",
    "accountStatus": "active",
    "telephoneNumber": "1-415-599-1100"
  }
}

```

This command changes John Doe's `ldapGroups` property in the OpenIDM 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 OpenDJ.

5. You can verify this change by querying John Doe's record on the LDAP server, as follows:

```

$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --request GET \
  "https://localhost:8443/openidm/system/ldap/account/uid=jdoe,ou=People,dc=example,dc=com"
{
  "_id": "uid=jdoe,ou=People,dc=example,dc=com",
  "telephoneNumber": "1-415-599-1100",
  "description": "Created for OpenIDM",
  "sn": "Doe",
  "dn": "uid=jdoe,ou=People,dc=example,dc=com",
  "ldapGroups": [
    "cn=openidm2,ou=Groups,dc=example,dc=com"
  ],
  "uid": "jdoe",
  "cn": "John Doe",
  "givenName": "John",
  "mail": "jdoe@example.com"
}

```

Procedure 3.6. To Run the Sample Using 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`.

The first time you log into the Admin UI, you are prompted to change your password. If you do not want to change your password at this time, simply click X to close this window, and continue with the sample.

2. Select the Mappings tab.

This tab shows two configured mappings, one from the `ldap` server to the OpenIDM repository (`managed/user`) and one from the OpenIDM repository to the `ldap` server.

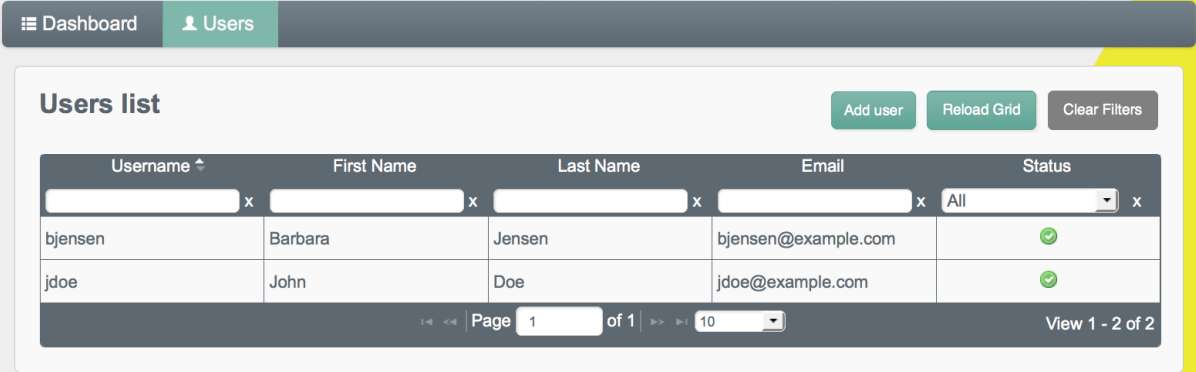
3. Click anywhere on the first mapping and click Reconcile Now.

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

4. Retrieve the users in the repository by clicking the User View link at the top right of the Admin UI.

This link opens the User View UI. If you did not change your password in the first step, you are prompted to change your password again. You can bypass this by simply clicking X to close the password prompt window.

5. Select the Users tab.



The screenshot shows the OpenIDM Users list interface. The navigation bar includes 'Dashboard' and 'Users' tabs. The 'Users list' section contains buttons for 'Add user', 'Reload Grid', and 'Clear Filters'. The table below lists two users:

Username	First Name	Last Name	Email	Status
bjensen	Barbara	Jensen	bjensen@example.com	✓
jdoe	John	Doe	jdoe@example.com	✓

The footer of the table indicates 'Page 1 of 1' and 'View 1 - 2 of 2'.

The two users from the LDAP server have been reconciled to the OpenIDM repository.

6. To retrieve the details of a specific user, click that username on the Users tab.

The following image shows the details of user **jdoe**.

John Doe's profile

Username [Change password](#)

First Name Last Name

Email address Role Administrator
 Tasks Manager
 User

Account status Mobile Phone

Address 1 Address 2

City Postal Code

Country State/Province

Linked Systems

Linked Resource

ldapGroups

item 1
cn=openidm,ou=Groups,dc=example,dc=com

Note the Linked Systems section of John Doe's profile. The Linked Resource item indicates the external resource on which John Doe's managed object is mapped, in this case, **ldap account**.

In this linked resource, John Doe's **ldapGroups** are displayed. Currently, John Doe is a member of **cn=openidm,ou=Groups,dc=example,dc=com**.

- Update John Doe's **ldapGroups** property, in the repository, to change his membership from the **openidm** group to the **openidm2** group.

In the current default version of the User View UI, only specific properties of a managed object can be edited. It is therefore not possible to update John Doe's **ldapGroups** property using the UI.

Instead, update his managed/user object over REST, as follows:

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '[
  {
    "operation": "replace",
    "field": "/ldapGroups",
    "value": ["cn=openidm2,ou=Groups,dc=example,dc=com"]
  }
]' \
"https://localhost:8443/openidm/managed/user?_action=patch&_queryId=for-username&uid=jdoe"
{
  "displayName": "John Doe",
  "userName": "jdoe",
  "stateProvince": "",
  "postalAddress": "",
  "effectiveAssignments": {},
  "ldapGroups": [
    "cn=openidm2,ou=Groups,dc=example,dc=com"
  ],
  "roles": [
    "openidm-authorized"
  ],
  "city": "",
  "effectiveRoles": [
    "openidm-authorized"
  ],
  "givenName": "John",
  "lastPasswordAttempt": "Wed Nov 26 2014 14:30:58 GMT+0200 (SAST)",
  "address2": "",
  "passwordAttempts": "0",
  "sn": "Doe",
  "mail": "jdoe@example.com",
  "country": "",
  "_rev": "2",
  "lastPasswordSet": "",
  "postalCode": "",
  "_id": "cec9e2d7-cb6c-4622-94a8-2cb43695476f",
  "description": "Created for OpenIDM",
  "accountStatus": "active",
  "telephoneNumber": "1-415-599-1100"
}
```

This command changes John Doe's `ldapGroups` property in the OpenIDM 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 OpenDJ.

8. You can verify this change by reloading John Doe's user profile in the User View UI, and looking at the value of his `ldapGroups` property in the Linked Systems panel.

The following image shows the updated record for John Doe, with a new `ldapGroups` value of `cn=openidm2,ou=Groups,dc=example,dc=com`.

John Doe's profile

Username [Change password](#)

First Name Last Name

Email address

Role Administrator
 Tasks Manager
 User

Account status

Mobile Phone

Address 1

Address 2

City

Postal Code

Country

State/Province

Linked Systems

Linked Resource

IdapGroups

item 1
cn=openidm2,ou=Groups,dc=example,dc=c

3.6. Sample 2d - Synchronizing LDAP Groups

Sample 2d also connects to an external LDAP server. This sample focuses on LDAP Group synchronization.

3.6.1. External LDAP Configuration

Configure the LDAP server as for sample 2, Section 3.3.1, "LDAP Server Configuration". The LDAP user must have write access to create users from OpenIDM on the LDAP server.

In addition, two LDAP Groups should be created, which can be found in the LDIF file: `openidm/samples/sample2d/data/Example.ldif` (if they have not already been added through sample 2c):

```
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
uniqueMember: uid=jdoe,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.

3.6.2. Install the Sample

Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM", then start OpenIDM with the configuration for sample 2d.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/sample2d
```

3.6.3. Running the Sample

The mapping configuration file (`sync.json`) for this sample includes three mappings:

- `systemLdapAccounts_managedUser`

Synchronizes users from the source LDAP server with the target OpenIDM repository,

- `managedUser_systemLdapAccounts`

Synchronizes changes from the OpenIDM repository to the LDAP server.

- `systemLdapGroups_managedGroup`

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

Due to the similarity with other OpenIDM samples, especially samples 2b and 2c, the focus of this sample is on the mapping unique to this sample, `systemLdapGroups_managedGroup`.

You can run this part of the sample by using the `curl` command-line utility, or by using the OpenIDM Administration UI. This section provides instructions for both methods.

Procedure 3.7. To Run the Sample Using the Command Line

1. Reconcile the repository over the REST interface for the group mapping, `systemLdapGroups_managedGroup` by running the following command:

```
$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --header "Content-Type: application/json" \
  --request POST \
  "https://localhost:8443/openidm/recon?
_action=recon&mapping=systemLdapGroups_managedGroup&waitForCompletion=true"
```

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

2. With the configuration of sample 2d, OpenIDM creates group objects from LDAP in OpenIDM. To list group objects by ID, run a query over the REST interface.

```
$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --request GET \
  "https://localhost:8443/openidm/managed/group?_queryFilter=true"
```

The resulting JSON object should include content similar to the following.

```
{
  "result" : [ {
    "dn" : "cn=openidm,ou=Groups,dc=example,dc=com",
    "_id" : "837df489-35d6-48d1-81a5-23792b49838a",
    "_rev" : "1",
    "description" : [ ],
    "uniqueMember" : [ "uid=jdoe,ou=People,dc=example,dc=com" ],
    "name" : [ "openidm" ]
  }, {
    "dn" : "cn=openidm2,ou=Groups,dc=example,dc=com",
    "_id" : "7575c1c7-86cf-43bc-bf1d-5c9cfc539124",
    "_rev" : "1",
    "description" : [ ],
    "uniqueMember" : [ "uid=bjensen,ou=People,dc=example,dc=com" ],
    "name" : [ "openidm2" ]
  } ],
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : -1
}
```

Procedure 3.8. To Run the Sample Using 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`.

The first time you log into the Admin UI, you are prompted to change your password. If you do not want to change your password at this time, click X to close this window, and continue with the sample.

2. Select the Mappings tab.

This tab shows three configured mappings, from the `ldap` server accounts repository to the OpenIDM repository (`managed/user`), from the OpenIDM repository back to the `ldap` server, and from the `ldap` server group accounts repository to the OpenIDM `managed/group` repository.

The screenshot shows the 'Mappings List' page in the Admin UI. At the top, there are tabs for 'Resources' and 'Mappings', with 'Mappings' selected. A 'New Mapping +' button is in the top right. Below the title, there are three mapping entries, each with a 'Reconciliation Status: Not yet Reconciled' and 'Delete' and 'Edit' buttons.

Mapping ID	Source	Target	Reconciliation Status	Actions
systemLdapAccounts_managedUser	Source system/ldap/account ldap	Target managed/user managed	Not yet Reconciled	Delete, Edit
managedUser_systemLdapAccounts	Source managed/user managed	Target system/ldap/account ldap	Not yet Reconciled	Delete, Edit
systemLdapGroups_managedGroup	Source system/ldap/group ldap	Target managed/group managed	Not yet Reconciled	Delete, Edit

3. Click anywhere on the third mapping and click Reconcile Now.

systemLdapGroups_managedGroup



Source
system/ldap/group
ldap





Target
managed/group
managed

▶ Status: Not yet Reconciled
[Reconcile Now](#)
[help](#)

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

4. Retrieve the groups in the repository by clicking the Correlation link below the mapping. Scroll down to Data Association Management.

Data Association Management

View: ALL SITUATIONS Change Source to Target Association Reconcile Selected Record

Source	Target
<input type="checkbox"/> cn=openidm2,ou=Groups,dc=example,dc=com uid=bjensen,ou=People,dc=example,dc=com openidm2	<input type="checkbox"/> cn=openidm2,ou=Groups,dc=example,dc=com uid=bjensen,ou=People,dc=example,dc=com openidm2
<input type="checkbox"/> cn=openidm,ou=Groups,dc=example,dc=com uid=jdoe,ou=People,dc=example,dc=com openidm	<input type="checkbox"/> cn=openidm,ou=Groups,dc=example,dc=com uid=jdoe,ou=People,dc=example,dc=com openidm

Page 1 of 10

The two groups from the LDAP server (source) have been reconciled to the OpenIDM repository (target).

3.7. Using the Groovy Connector Toolkit to Create Scripted Connectors

OpenICF 1.4 introduces a generic Groovy Connector Toolkit that enables you to run Groovy scripts on any external resource.

The Groovy Connector Toolkit is not a complete connector, in the traditional sense. Rather, it is a framework within which you must write your own Groovy scripts to address the requirements of your implementation. Specific scripts are provided within these samples, which demonstrate how the

Groovy Connector Toolkit can be used. These scripts cannot be used "as is" in your deployment, but are a good starting point on which to base your customization.

3.7.1. Sample 3 - Using the Groovy Connector Toolkit to Connect to MySQL With ScriptedSQL

This sample uses a ScriptedSQL implementation of the Groovy Connector Toolkit in which an instance of the ScriptedSQL framework is made to work with the `hrdb` schema. The "connector" in this sample is therefore really an `hrdb` connector that is implemented with the help of the Groovy Connector Toolkit.

The Groovy Connector Toolkit is bundled with OpenIDM 3.1, in the JAR `openidm/connectors/groovy-connector-1.4.1.0.jar`. The connector configuration file for this sample (`sample3/conf/provisioner.openicf-scriptedsql.json`) indicates the ScriptedSQL implementation of the Groovy connector as follows:

```
{
  "name" : "scriptedsql",
  "connectorRef" : {
    "bundleName" : "org.forgerock.openicf.connectors.groovy-connector",
    "bundleVersion" : "[1.4.0.0,2.0.0.0)",
    "connectorName" : "org.forgerock.openicf.connectors.scriptedsql.ScriptedSQLConnector"
  },
  ...
}
```

This sample demonstrates the use of *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, 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 those data types. The sample defines one complex data type, `cars`, discussed in more detail later in this section.

The mapping in this sample shows the use of event hooks to perform an action. The mapping configuration from the internal repository to the external `hrdb` database system (`managedUser_systemHrdb`), defined in the `sync.json` file, includes two script hooks. The first is for an `onCreate` event and the second for an `onUpdate` event. For both events, OpenIDM logs a statement to the log when a user is created or updated in the external system. The script source is included in the mapping, but can also be called from an external file. For more information, see Appendix F, "Scripting Reference" in the *Integrator's Guide*.

This sample also demonstrates the configuration of custom scripted endpoints, defined in the connector configuration (`provisioner.openicf-scriptedsql.json`).

Caution

Because MySQL cannot "un-hash" user passwords there is no way for a reconciliation operation to retrieve and store 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 from managed/user will be created by the 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.

This sample configuration assumes a MySQL server, running on the localhost, and listening on port 3306. The sample database, `hrdb`, is created over REST.

The Groovy scripts required for the sample are located in the `sample3/tools` directory. You will need to 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.

Note

This sample is supported with Java version 7 only. The dependencies required for the sample are provided in the `openidm/bundle` folder and do not require a separate download. If you are using the ScriptedSQL connector implementation with Java version 6, you must download the related dependencies. For more information, see Section 3.7.1.5, "Using Sample 3 With Java Version 6" at the end of this section.

Prepare a fresh installation of OpenIDM before trying this sample.

3.7.1.1. Setting Up the Sample

This sample requires an installed, running MySQL instance. The connection to the MySQL server is defined in the connector configuration file (`sample3/conf/provisioner.openicf-scriptedsql.json`). The default connector configuration file assumes the following connection details:

```
"configurationProperties" : {  
  "username" : "root",  
  "password" : "password",  
  "driverClassName" : "com.mysql.jdbc.Driver",  
  "url" : "jdbc:mysql://localhost:3306/hrdb",
```

As indicated in the preceding excerpt of the connector configuration file, the sample expects the following MySQL configuration:

- The database is available on the localhost.
- The database listens on port 3306.
- You can connect over the network to the database with user `root` and password `password`.

If you have an existing MySQL instance, running on a different host, or port, adjust the connector configuration accordingly.

1. Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM".
2. Download the MySQL Driver, (MySQL Connector/J, version 5.1 or later) from the MySQL website. Unpack the download and copy the `.jar` into the `openidm/bundle` directory.

```
$ cp mysql-connector-java-version-bin.jar /path/to/openidm/bundle/
```

3. Create an empty **hrdb** database.

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.25a MySQL Community Server (GPL)
mysql> CREATE DATABASE hrdb CHARACTER SET utf8 COLLATE utf8_bin;
Query OK, 1 row affected (0.00 sec)
mysql> quit
Bye
```

4. Start OpenIDM with the configuration for sample 3.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/sample3
Executing ./startup.sh...
Using OPENIDM_HOME: /path/to/openidm
Using PROJECT_HOME: /path/to/openidm/samples/sample3/
Using OPENIDM_OPTS: -Xmx1024m -Xms1024m
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/samples/sample3/
                        /conf/logging.properties
Using boot properties at /path/to/openidm/samples/sample3/conf/boot/boot.properties
OpenIDM version "3.1" (revision: 0) null
null
-> OpenIDM ready
```

5. Populate the MySQL database with sample data.

This step executes a custom script, over the REST interface, that resets and populates the **hrdb** database.

The script is referenced in a system action, defined in the connector configuration file ([conf/provisioner.openicf-scriptedsql.json](#)) as follows:

```
"systemActions" : [
  {
    "scriptId" : "ResetDatabase",
    "actions" : [
      {
        "systemType" : ".*ScriptedSQLConnector",
        "actionType" : "Groovy",
        "actionFile" : "tools/ResetDatabaseScript.groovy" }
    ]
  }
  ...
]
```

You can run this script again, at any point, to reset the database. Currently, only Groovy script is supported for these types of actions.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/system/scriptedsql?action=script&scriptId=ResetDatabase"
{
  "actions": [
    {
      "result": "Database reset successful."
    }
  ]
}
```

The **hrdb** database should now be populated with sample data.

You can review the contents of the database as follows:

```
$ mysql -u root -p
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, as they cannot be read into OpenIDM as clear text. The SHA-1 Hash function is used for compatibility reasons. Use a more secure algorithm in a production database.

3.7.1.2. Reconciling the Repository

1. The mapping configuration file (**sync.json**) for this sample includes the mapping **systemHrdb_managedUser**, which synchronize users from the source **hrdb** database with the target OpenIDM repository.

You can test this part of the sample by using the `curl` command-line utility, or the OpenIDM Administration UI.

- 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`.

The first time you log into the Admin UI, you are prompted to change your password. If you do not want to change your password at this time, simply click X to close this window, and continue with the sample.

- Select the Mappings tab.

This tab shows two configured mappings, one from the `hrdb` database to the OpenIDM repository (`managed/user`), and one in the opposite direction.

The screenshot shows the 'Mappings' tab in the OpenIDM Administration UI. The 'Mapping List' section displays two mappings:

Source	Target	Sync Status	Actions
<code>system/scriptedsql/account</code> <small>scriptedsql</small>	<code>managed/user</code> <small>managed</small>	Not yet synced	Delete, Edit
<code>managed/user</code> <small>managed</small>	<code>system/scriptedsql/account</code> <small>scriptedsql</small>	Not yet synced	Delete, Edit

- Click on the first mapping (the mapping from `system/scriptedsql/account` to `managed/user`).
 - Click Reconcile Now.
- To reconcile the repository by using the command-line, launch the reconciliation operation with the following command:

```

$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/recon?
_action=recon&mapping=systemHrdb_managedUser&waitForCompletion=true"
{
  "state": "SUCCESS",
  "_id": "f3c618aa-cc3b-49ed-9a3a-00b012db2513"
}

```

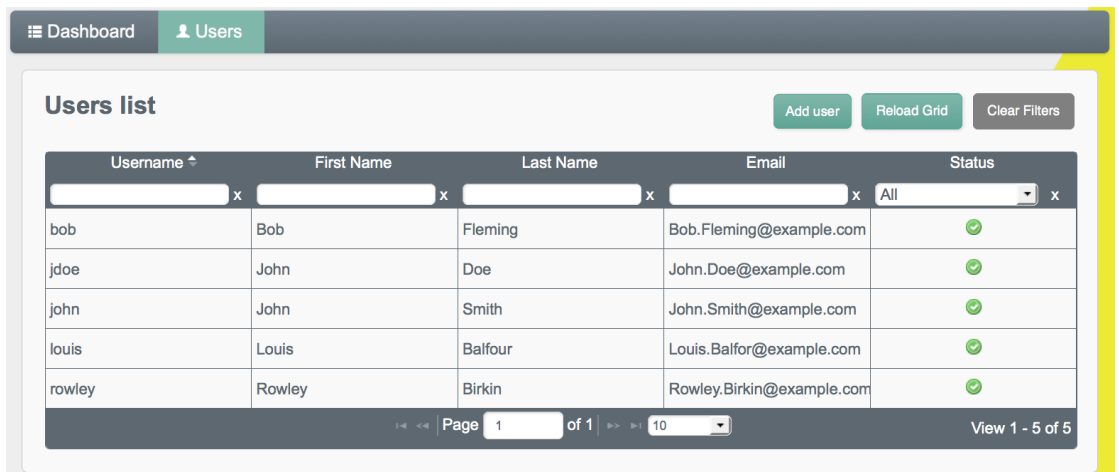
The reconciliation operation creates the five users from the MySQL database in the OpenIDM repository.

2. Retrieve the list of users from the repository.

- To retrieve the users in the repository by using the Administration UI:
 1. Click the User View link at the top right of the Admin UI.

This opens the regular OpenIDM UI. If you did not change your password in the first step, you are prompted to change your password again. You can bypass this by simply clicking X to close the password prompt window.

2. Select the Users tab.



The five users from the **hrdb** database have been reconciled to the OpenIDM repository.

- To retrieve the details of a specific user, click that username on the Users tab.

The following image shows the details for the user Rowley Birkin.

Rowley Birkin's profile

Username: [Change password](#)

First Name: Last Name:

Email address: Role: Administrator, Tasks Manager, User

Account status: Mobile Phone:

Address 1: Address 2:

City: Postal Code:

Country: State/Province:

Linked Systems

Linked Resource:

organization: SALES

cars

Collapse

item 1

Collapse

year: 2013

make: BMW

model: 328ci

- To retrieve the users from the repository by using the command-line, query the IDs in the repository as follows:

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/managed/user?_queryId=query-all-ids"
{
  "remainingPagedResults": -1,
```

```

"pagedResultsCookie": null,
"resultCount": 5,
"result": [
  {
    "_rev": "0",
    "_id": "9ee75d24-899c-4221-97a0-0aca298febd6"
  },
  {
    "_rev": "0",
    "_id": "25be1c4c-0c2a-48b6-96f6-58d2e4d2357d"
  },
  {
    "_rev": "0",
    "_id": "2850c77b-f51a-4fb2-8cc4-4c1d03108ac2"
  },
  {
    "_rev": "0",
    "_id": "126d74e1-1c03-4774-b18d-bd4d1dfdf884"
  },
  {
    "_rev": "0",
    "_id": "46570045-0644-45c6-af10-c88ad3df93cc"
  }
]
}

```

To retrieve a complete user record, query for an individual user, by his userName. The following example returns the record for the user **Rowley Birkin**.

```

$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --request GET \
  "https://localhost:8443/openidm/managed/user/?_queryId=for-username&uid=rowley"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 1,
  "result": [
    {
      "stateProvince": "",
      "userName": "rowley",
      "postalAddress": "",
      "effectiveAssignments": null,
      "roles": [
        "openidm-authorized"
      ],
      "telephoneNumber": "",
      "country": "",
      "effectiveRoles": [
        "openidm-authorized"
      ],
      "givenName": "Rowley",
      "address2": "",
      "lastPasswordAttempt": "Wed Nov 05 2014 10:02:02 GMT+0200 (SAST)",
      "passwordAttempts": "0",
      "sn": "Birkin",
    }
  ]
}

```

```
"mail": "Rowley.Birkin@example.com",
"city": "",
"lastPasswordSet": "",
"organization": "SALES",
"postalCode": "",
"_id": "126d74e1-1c03-4774-b18d-bd4d1dfdf884",
"_rev": "1",
"cars": [
  {
    "model": "328ci",
    "year": "2013",
    "make": "BMW"
  },
  {
    "model": "ES300",
    "year": "2010",
    "make": "Lexus"
  }
],
"accountStatus": "active"
}
]
```

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 OpenIDM can understand. With support for complex objects, the data is passed through to OpenIDM as a list of `car` objects. Data is synchronized from OpenIDM 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`). OpenIDM does not maintain group membership in this way, so the Groovy scripts do the work to translate membership between the two resources.

3.7.1.3. Using Paging With Sample 3

All OpenICF 1.4.1.0 connectors 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 \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
'https://localhost:8443/openidm/system/scriptedsql/account?_queryFilter=uid+sw+""&_fields=id
,uid&_pageSize=2&_sortKeys=timestamp,id'
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": "2014-09-10 12:18:46.0,2",
  "resultCount": 2,
  "result": [
    {
      "_id": "1",
      "uid": "bob"
    },
    {
      "_id": "2",
      "uid": "rowley"
    }
  ]
}
```

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. You must URL-encode the space character between the date and time in the `"pagedResultsCookie"` value, with `%20`, as shown in the following example:

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
'https://localhost:8443/openidm/system/scriptedsql/account?_queryId=query-all-
ids&_pageSize=2&_sortKeys=timestamp,id&_pagedResultsCookie2014-09-10%2012:18:46.0,2"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": "2014-09-10 12:18:46.0,4",
  "resultCount": 2,
  "result": [
    {
      "_id": "3",
      "uid": "louis"
    },
    {
      "_id": "4",
      "uid": "john"
    }
  ]
}
```

For more information about paging support, see Section 7.3.5, "Paging Query Results" in the *Integrator's Guide*.

3.7.1.4. Reconciling the MySQL Database

The second mapping defined in the mapping configuration file for this sample (`managedUser_systemHrdb`) synchronizes users from the source OpenIDM repository to the target `hrdb` database. Instead of running a manual reconciliation operation to synchronize these resources, edit one of the user records in the OpenIDM UI.

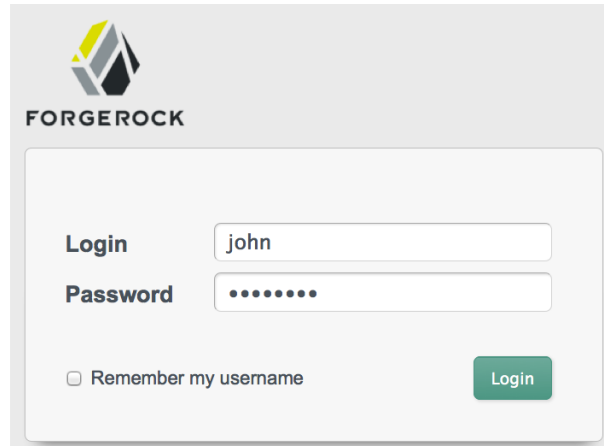
1. Log in to the OpenIDM UI (<https://localhost:8443/openidmui>) as any one of the users that was created in the repository by the previous reconciliation operation.

The initial clear text passwords of these users can be seen in the `sample3/tools/ResetDatabaseScript.groovy` script and are provided here for ease of reference:

```
"bob", "password1"  
"rowley", "password2"  
"louis", "password3"  
"john", "password4"  
"jdoe", "password5"
```

Regular users can update their profiles or passwords. Any changes are automatically synchronized back to the `hrdb` database, through the implicit synchronization mechanism.

For example, log into the UI as user `john` with the password `password4`.



The screenshot shows the OpenIDM UI login page. At the top left is the ForgeRock logo. Below it, the text "FORGEROCK" is displayed. The main content area contains a login form with the following elements:

- A "Login" label next to a text input field containing the username "john".
- A "Password" label next to a password input field containing masked characters ".....".
- A checkbox labeled "Remember my username" which is currently unchecked.
- A green "Login" button.

2. Update the password or profile of the user `john` (or the user as whom you logged in in the previous step).

The following example updates the email address for user `john` to `John.P.Smith@example.com`.

User profile

Username <input type="text" value="john"/>	Change Security Data
First Name <input type="text" value="John"/>	Last Name <input type="text" value="Smith"/>
Email address <input type="text" value="John.P.Smith@example.com"/>	Mobile Phone <input type="text"/>
Address 1 <input type="text"/>	Address 2 <input type="text"/>
City <input type="text"/>	Postal Code <input type="text"/>
Country <input type="text" value="Please Select"/>	State/Province <input type="text"/>

3. Changes are automatically synchronized to the `hrdb` database. Retrieve user `john`'s record from the database by including his `_id` in a query on the system endpoint, for example:

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/scriptedsql/account/4"
{
  "_id": "4",
  "email": "John.P.Smith@example.com",
  "organization": "SUPPORT",
  "firstName": "John",
  "lastName": "Smith",
  "uid": "john",
  "fullName": "John Smith"
}
```

3.7.1.5. Using Sample 3 With Java Version 6

The default sample works only with Java Version 7, and bundles the required dependencies in the `openidm/bundle` folder. To use the sample with Java Version 6, you must remove these dependencies and download the dependencies specific to Java 6. Follow these steps:

1. Remove the following jar files from the `openidm/bundle` folder:

- `tomcat-juli-8.0.12.jar`
- `tomcat-jdbc-8.0.12.jar`

2. Create a `lib` folder in the OpenIDM installation directory.

```
$ cd /path/to/openidm
$ mkdir lib
```

3. Download the following dependencies to the `openidm/lib` directory:

- Apache Geronimo Tomcat fork (Juli), `juli-7.0.39.2.jar` (org.apache.geronimo.ext.tomcat:juli:7.0.39.2)
- Tomcat JDBC Pool Package, `tomcat-jdbc-7.0.53.jar` (org.apache.tomcat:tomcat-jdbc:7.0.53)

The remainder of the sample should work as for Java Version 7.

3.7.2. Sample - Using the Groovy Connector Toolkit to Connect to OpenDJ With ScriptedREST

This sample uses the Groovy Connector Toolkit to implement a ScriptedREST connector, which interacts with the OpenDJ REST API.

The Groovy Connector Toolkit is bundled with OpenIDM 3.1, in the JAR `openidm/connectors/groovy-connector-1.4.1.0.jar`.

The connector configuration file for this sample (`samples/scriptedrest2dj/conf/provisioner.openicf-scriptedrest.json`) indicates the ScriptedREST implementation of the Groovy connector as follows:

```
{
  "name": "scriptedrest",
  "connectorRef": {
    "connectorHostRef": "#LOCAL",
    "connectorName": "org.forgerock.openicf.connectors.scriptedrest.ScriptedRESTConnector",
    "bundleName": "org.forgerock.openicf.connectors.groovy-connector",
    "bundleVersion": "[1.4.0.0,2.0.0.0)"
  },
  ...
}
```

The Groovy scripts required for the sample are located in the `samples/scriptedrest2dj/tools` directory. You will need to 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.

3.7.2.1. Setting Up OpenDJ

This sample assumes an OpenDJ server, running on the localhost. Follow these steps to install and configure an OpenDJ instance.

1. Download and extract the OpenDJ zip archive from the [OpenDJ Project page](#) .
2. Install OpenDJ using the command-line setup, as follows:

```
$ cd /path/to/opendj
$ ./setup --cli \
--hostname localhost \
--ldapPort 1389 \
--rootUserDN "cn=Directory Manager" \
--rootUserPassword password \
--adminConnectorPort 4444 \
--addBaseEntry \
--baseDN dc=com \
--acceptLicense \
--no-prompt
...
Configuring Directory Server ..... Done.
Creating Base Entry dc=com ..... Done.
Starting Directory Server ..... Done
.
...
```

The sample assumes the following configuration:

- The server is installed on the localhost.
- The server listens for LDAP connections on port 1389.
- The administration connector port is 4444.
- The root user DN is `cn=Directory Manager`.
- The root user password is `password`.

3. Configure the OpenDJ server for replication.

To enable LiveSync, this server must be configured for replication, even if it does not actually participate in a replication topology. The following commands configure the server for replication.

```

$ cd /path/to/openssl/bin
$ ./dsconfig create-replication-server \
--hostname localhost \
--port 4444 \
--bindDN "cn=Directory Manager" \
--bindPassword password \
--provider-name "Multimaster Synchronization" \
--set replication-port:8989 \
--set replication-server-id:2 \
--type generic \
--trustAll \
--no-prompt

$ ./dsconfig create-replication-domain \
--hostname localhost \
--port 4444 \
--bindDN "cn=Directory Manager" \
--bindPassword password \
--provider-name "Multimaster Synchronization" \
--domain-name example_com \
--set base-dn:dc=example,dc=com \
--set replication-server:localhost:8989 \
--set server-id:3 \
--type generic \
--trustAll \
--no-prompt

```

4. Enable HTTP access to the OpenDJ directory server as follows:

```

$ ./dsconfig set-connection-handler-prop \
--hostname localhost \
--port 4444 \
--bindDN "cn=Directory Manager" \
--bindPassword password \
--handler-name "HTTP Connection Handler" \
--set enabled:true \
--set listen-port:8090 \
--no-prompt \
--trustAll

```

5. Enable the OpenDJ HTTP access log.

```

$ ./dsconfig set-log-publisher-prop \
--hostname localhost \
--port 4444 \
--bindDN "cn=Directory Manager" \
--bindPassword password \
--publisher-name "File-Based HTTP Access Logger" \
--set enabled:true \
--no-prompt \
--trustAll

```

6. Import the LDIF data required for the sample.

```
$ ./ldapmodify \
  --bindDN "cn=Directory Manager" \
  --bindPassword password \
  --hostname localhost \
  --port 1389 \
  --filename /path/to/openidm/samples/scriptedrest2dj/data/ldap.ldif
Processing ADD request for dc=example,dc=com
ADD operation successful for DN dc=example,dc=com
Processing ADD request for ou=Administrators,dc=example,dc=com
ADD operation successful for DN ou=Administrators,dc=example,dc=com
Processing ADD request for uid=idm,ou=Administrators,dc=example,dc=com
ADD operation successful for DN uid=idm,ou=Administrators,dc=example,dc=com
Processing ADD request for ou=People,dc=example,dc=com
ADD operation successful for DN ou=People,dc=example,dc=com
Processing ADD request for ou=Groups,dc=example,dc=com
ADD operation successful for DN ou=Groups,dc=example,dc=com
```

- To configure the mapping between JSON resources and LDAP entries, copy the the configuration file for the HTTP connection handler (`/path/to/openidm/samples/scriptedrest2dj/data/http-config.json`) to OpenDJ's configuration directory.

```
$ cd /path/to/opendj
$ cp /path/to/openidm/samples/scriptedrest2dj/data/http-config.json config/
```

- Restart OpenDJ for the configuration change to take effect.

```
$ cd /path/to/opendj/bin
$ ./stop-ds --restart
Stopping Server...
The Directory Server has started successfully
```

OpenDJ is now configured for this sample.

3.7.2.2. Running the Sample

This section illustrates the basic CRUD operations on users and groups using the ScriptedREST connector and the OpenDJ REST API. Note that the power of the Groovy connector is in the associated Groovy scripts, and their application in your specific deployment. The scripts provided with this sample are specific to the sample and customization of the scripts is required.

- Start OpenIDM with the configuration for the ScriptedREST sample.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/scriptedrest2dj/
```

- Check the connector configuration is correct by obtaining the status of the connector, over REST.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/system/scriptedrest?_action=test"
{
  "ok": true,
  "connectorRef": {
    "bundleVersion": "[1.4.0.0,2.0.0.0)",
    "bundleName": "org.forgerock.openicf.connectors.groovy-connector",
    "connectorName": "org.forgerock.openicf.connectors.scriptedrest.ScriptedRESTConnector"
  },
  "objectTypes": [
    "group",
    "account"
  ],
  "config": "config/provisioner.openicf/scriptedrest",
  "enabled": true,
  "name": "scriptedrest"
}
```

3. Create a group entry on the OpenDJ server.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "_id" : "group1"
}' \
"https://localhost:8443/openidm/system/scriptedrest/group?_action=create"
{
  "_id": "group1",
  "cn": "group1",
  "members": null,
  "lastModified": null,
  "created": "2014-09-24T17:34:27Z",
  "displayName": "group1"
}
```

4. Create a user entry on the OpenDJ server.


```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "givenName" : "Steven",
  "familyName" : "Carter",
  "emailAddress" : "scarter@example.com",
  "telephoneNumber" : "444-444-4444",
  "password" : "Passw0rd",
  "displayName" : "Steven.Carter",
  "uid" : "scarter"
}' \
https://localhost:8443/openidm/system/scriptedrest/account?action=create
{
  "_id": "scarter",
  "displayName": "Steven.Carter",
  "uid": "scarter",
  "groups": null,
  "familyName": "Carter",
  "emailAddress": "steven.carter@example.com",
  "givenName": "Steven",
  "created": "2014-09-24T17:35:46Z",
  "telephoneNumber": "444-444-4444"
}
```

Notice that at this stage, the user is not a member of any group.

5. Update Steven Carter's entry, by modifying his telephone number.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "givenName" : "Steven",
  "familyName" : "Carter",
  "emailAddress" : "scarter@example.com",
  "telephoneNumber" : "555-555-5555",
  "password" : "Passw0rd",
  "displayName" : "Steven.Carter",
  "uid" : "scarter"
}' \
https://localhost:8443/openidm/system/scriptedrest/account/scarter
{
  "_id": "scarter",
  "displayName": "Steven.Carter",
  "uid": "scarter",
  "groups": null,
  "familyName": "Carter",
  "emailAddress": "steven.carter@example.com",
  "givenName": "Steven",
  "created": "2014-09-24T17:35:46Z",
  "telephoneNumber": "555-555-5555"
}
```

6. Add Steven Carter to the group you created previously, by updating the group entry.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "_id" : "group1",
  "members" : [{"_id" : "scarter"}]
}' \
https://localhost:8443/openidm/system/scriptedrest/group/group1
{
  "_id": "group1",
  "cn": "group1",
  "members": [
    {
      "displayName": "Steven.Carter",
      "_id": "scarter"
    }
  ],
  "lastModified": "2014-09-24T17:31:42Z",
  "created": "2014-09-24T17:27:37Z",
  "displayName": "group1"
}
```

7. Read Steven Carter's entry, to verify that he is now a member of group1.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
https://localhost:8443/openidm/system/scriptedrest/account/scarter
{
  "_id": "scarter",
  "displayName": "Steven.Carter",
  "uid": "scarter",
  "groups": [
    {
      "_id": "group1"
    }
  ],
  "familyName": "Carter",
  "emailAddress": "steven.carter@example.com",
  "givenName": "Steven",
  "created": "2014-09-24T17:31:04Z",
  "telephoneNumber": "555-555-5555"
}
```

8. Read the group entry to verify its members.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
https://localhost:8443/openidm/system/scriptedrest/group/group1
{
  "_id": "group1",
  "cn": "group1",
  "members": [
    {
      "displayName": "Steven.Carter",
      "_id": "scarter"
    }
  ],
  "lastModified": "2014-09-24T17:31:42Z",
  "created": "2014-09-24T17:27:37Z",
  "displayName": "group1"
}
```

9. Delete the user and group entries, returning the OpenDJ server to its initial state.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request DELETE \
https://localhost:8443/openidm/system/scriptedrest/account/scarter
{
  "_id": "scarter"
}
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request DELETE \
https://localhost:8443/openidm/system/scriptedrest/group/group1
{
  "_id": "group1"
}
```

3.7.3. Sample - Using the Groovy Connector Toolkit to Connect to OpenDJ With ScriptedCREST

This sample uses the Groovy Connector Toolkit to implement a ScriptedCREST connector, which interacts with the ForgeRock Commons REST (CREST) API to connect to an OpenDJ instance. The main difference between a CREST-based API and a generic REST API is that the CREST API is inherently recognizable by all ForgeRock products. As such, the sample can leverage CREST resources in the groovy scripts, to create CREST requests.

The Groovy Connector Toolkit is bundled with OpenIDM 3.1, in the JAR `openidm/connectors/groovy-connector-1.4.1.0.jar`.

The connector configuration file for this sample (`samples/scriptedcrest2dj/conf/provisioner.openicf-scriptedcrest.json`) indicates the ScriptedCREST implementation of the Groovy Connector Toolkit as follows:

```
{
  "name": "scriptedcrest",
  "connectorRef": {
    "connectorHostRef": "#LOCAL",
    "connectorName": "org.forgerock.openicf.connectors.scriptedcrest.ScriptedCRESTConnector",
    "bundleName": "org.forgerock.openicf.connectors.groovy-connector",
    "bundleVersion": "[1.4.0.0,2.0.0.0)"
  },
  ...
}
```

The Groovy scripts required for the sample are located in the `samples/scriptedcrest2dj/tools` directory. You will need to 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.

3.7.3.1. Setting Up OpenDJ

This sample assumes an OpenDJ server, running on the localhost. Follow these steps to install and configure an OpenDJ instance.

1. Download and extract the OpenDJ zip archive from the [OpenDJ Project page](#).
2. Install OpenDJ using the command-line setup, as follows:

```
$ cd /path/to/opendj
$ ./setup --cli \
  --hostname localhost \
  --ldapPort 1389 \
  --rootUserDN "cn=Directory Manager" \
  --rootUserPassword password \
  --adminConnectorPort 4444 \
  --addBaseEntry \
  --baseDN dc=com \
  --acceptLicense \
  --no-prompt
...
Configuring Directory Server ..... Done.
Creating Base Entry dc=com ..... Done.
Starting Directory Server ..... Done
.
...
```

The sample assumes the following configuration:

- The server is installed on the localhost.
 - The server listens for LDAP connections on port 1389.
 - The administration connector port is 4444.
 - The root user DN is `cn=Directory Manager`.
 - The root user password is `password`.
3. Configure the OpenDJ server for replication.

To enable liveSync, this server must be configured for replication, even if it does not actually participate in a replication topology. The following commands configure the server for replication.

```

$ cd /path/to/opendj/bin
$ ./dsconfig create-replication-server \
--hostname localhost \
--port 4444 \
--bindDN "cn=Directory Manager" \
--bindPassword password \
--provider-name "Multimaster Synchronization" \
--set replication-port:8989 \
--set replication-server-id:2 \
--type generic \
--trustAll \
--no-prompt

$ ./dsconfig create-replication-domain \
--hostname localhost \
--port 4444 \
--bindDN "cn=Directory Manager" \
--bindPassword password \
--provider-name "Multimaster Synchronization" \
--domain-name example_com \
--set base-dn:dc=example,dc=com \
--set replication-server:localhost:8989 \
--set server-id:3 \
--type generic \
--trustAll \
--no-prompt

```

4. Enable HTTP access to the OpenDJ directory server as follows:

```

$ ./dsconfig set-connection-handler-prop \
--hostname localhost \
--port 4444 \
--bindDN "cn=Directory Manager" \
--bindPassword password \
--handler-name "HTTP Connection Handler" \
--set enabled:true \
--set listen-port:8090 \
--no-prompt \
--trustAll

```

5. Enable the OpenDJ HTTP access log.

```

$ ./dsconfig set-log-publisher-prop \
--hostname localhost \
--port 4444 \
--bindDN "cn=Directory Manager" \
--bindPassword password \
--publisher-name "File-Based HTTP Access Logger" \
--set enabled:true \
--no-prompt \
--trustAll

```

6. Import the LDIF data required for the sample.

```
$ ./ldapmodify \
--bindDN "cn=Directory Manager" \
--bindPassword password \
--hostname localhost \
--port 1389 \
--filename /path/to/openidm/samples/scriptedcrest2dj/data/ldap.ldif
Processing ADD request for dc=example,dc=com
ADD operation successful for DN dc=example,dc=com
Processing ADD request for ou=Administrators,dc=example,dc=com
ADD operation successful for DN ou=Administrators,dc=example,dc=com
Processing ADD request for uid=idm,ou=Administrators,dc=example,dc=com
ADD operation successful for DN uid=idm,ou=Administrators,dc=example,dc=com
Processing ADD request for ou=People,dc=example,dc=com
ADD operation successful for DN ou=People,dc=example,dc=com
Processing ADD request for ou=Groups,dc=example,dc=com
ADD operation successful for DN ou=Groups,dc=example,dc=com
```

- To configure the mapping between JSON resources and LDAP entries, copy the the configuration file for the HTTP connection handler (`/path/to/openidm/samples/scriptedcrest2dj/data/http-config.json`) to OpenDJ's configuration directory.

```
$ cd /path/to/opendj
$ cp /path/to/openidm/samples/scriptedcrest2dj/data/http-config.json config/
```

- Restart OpenDJ for the configuration change to take effect.

```
$ cd /path/to/opendj/bin
$ ./stop-ds --restart
Stopping Server...
The Directory Server has started successfully
```

OpenDJ is now configured for this sample.

3.7.3.2. Running the Sample

This section illustrates the basic CRUD operations on users and groups using the ScriptedCREST connector implementation and the OpenDJ REST API. Note that the power of the Groovy connector is in the associated Groovy scripts, and their application in your specific deployment. The scripts provided with this sample are specific to the sample and customization of the scripts is required.

- Start OpenIDM with the configuration for the ScriptedCREST sample.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/scriptedcrest2dj/
```

- Check the connector configuration is correct by obtaining the status of the connector, over REST.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/system/scriptedcrest?_action=test"
{
  "ok": true,
  "connectorRef": {
    "bundleVersion": "[1.4.0.0,2.0.0.0)",
    "bundleName": "org.forgerock.openicf.connectors.groovy-connector",
    "connectorName": "org.forgerock.openicf.connectors.scriptedcrest.ScriptedCRESTConnector"
  },
  "objectTypes": [
    "groups",
    "users"
  ],
  "config": "config/provisioner.openicf/scriptedcrest",
  "enabled": true,
  "name": "scriptedcrest"
}
```

3. Create a group entry on the OpenDJ server.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "_id" : "group1"
}' \
"https://localhost:8443/openidm/system/scriptedcrest/groups?_action=create"
{
  "_rev": "0000000028f53bdf",
  "_id": "group1",
  "displayName": "group1",
  "meta": {
    "created": "2014-10-17T07:43:13Z"
  }
}
```

4. Create a user entry on the OpenDJ server.

```
$ curl \
--cacert self-signed.crt \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request POST \
--data '{
  "name": {
    "familyName": "Carter",
    "givenName" : "Steven"
  }
},'
```



```

    "contactInformation": {
      "emailAddress" : "scarter@example.com",
      "telephoneNumber" : "444-444-4444"
    },
    "password" : "TestPassw0rd",
    "displayName" : "Steven.Carter",
    "_id" : "scarter"
  }' \
  "https://localhost:8443/openidm/system/scriptedcrest/users?action=create"
{
  "_rev": "00000000d84482de",
  "meta": {
    "created": "2014-10-17T08:07:46Z"
  },
  "userName": "scarter@example.com",
  "contactInformation": {
    "emailAddress": "scarter@example.com",
    "telephoneNumber": "444-444-4444"
  },
  "name": {
    "givenName": "Steven",
    "familyName": "Carter"
  },
  "displayName": "Steven.Carter",
  "_id": "scarter"
}

```

Notice that at this stage, the user is not a member of any group.

5. Update Steven Carter's entry, by modifying his telephone number.

```

$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "name": {
    "familyName": "Carter",
    "givenName": "Steven"
  },
  "contactInformation": {
    "emailAddress" : "scarter@example.com",
    "telephoneNumber" : "555-555-5555"
  },
  "password" : "TestPassw0rd",
  "displayName" : "Steven.Carter",
  "_id" : "scarter"
}' \
  "https://localhost:8443/openidm/system/scriptedcrest/users/scarter"
{
  "_rev": "00000000eb8ba31c",
  "meta": {
    "created": "2014-10-17T08:07:46Z",
    "lastModified": "2014-10-17T08:25:05Z"
  },
  "userName": "scarter@example.com",

```

```

"contactInformation": {
  "emailAddress": "scarter@example.com",
  "telephoneNumber": "555-555-5555"
},
"name": {
  "givenName": "Steven",
  "familyName": "Carter"
},
"displayName": "Steven.Carter",
"_id": "scarter"
}

```

6. Add Steven Carter to the group you created previously, by updating the members of the group entry.

```

$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "_id": "group1",
  "members": [{"_id": "scarter"}]
}' \
"https://localhost:8443/openidm/system/scriptedcrest/groups/group1"
{
  "_rev": "0000000011ed6ea1",
  "members": [
    {
      "displayName": "Steven.Carter",
      "_id": "scarter"
    }
  ],
  "_id": "group1",
  "displayName": "group1",
  "meta": {
    "created": "2014-10-17T07:43:13Z",
    "lastModified": "2014-10-17T08:26:41Z"
  }
}

```

7. Read Steven Carter's entry, to verify that he is now a member of group1.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/scriptedcrest/users/scarter"
{
  "_rev": "00000000eb8ba31c",
  "groups": [
    {
      "_id": "group1"
    }
  ],
  "meta": {
    "created": "2014-10-17T08:07:46Z",
    "lastModified": "2014-10-17T08:25:05Z"
  },
  "userName": "scarter@example.com",
  "contactInformation": {
    "emailAddress": "scarter@example.com",
    "telephoneNumber": "555-555-5555"
  },
  "name": {
    "givenName": "Steven",
    "familyName": "Carter"
  },
  "displayName": "Steven.Carter",
  "_id": "scarter"
}
```

8. Read the group entry to verify its members.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/scriptedcrest/groups/group1"
{
  "_rev": "0000000011ed6ea1",
  "members": [
    {
      "displayName": "Steven.Carter",
      "_id": "scarter"
    }
  ],
  "_id": "group1",
  "displayName": "group1",
  "meta": {
    "created": "2014-10-17T07:43:13Z",
    "lastModified": "2014-10-17T08:26:41Z"
  }
}
```

9. Delete the user and group entries, returning the OpenDJ server to its initial state.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request DELETE \
"https://localhost:8443/openidm/system/scriptedcrest/users/scarter"
{
  "_id": "scarter"
}
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request DELETE \
"https://localhost:8443/openidm/system/scriptedcrest/groups/group1"
{
  "_id": "group1"
}
```

3.7.4. Sample - Using the Groovy Connector Toolkit to Connect to Microsoft Azure AD

This sample uses the Groovy Connector Toolkit to implement a Microsoft Azure AD connector, which interacts with the ForgeRock Commons REST (CREST) API to connect to an Azure-based instance of Active Directory. The main difference between a CREST-based API and a generic REST API is that the CREST API is inherently recognizable by all ForgeRock products. As such, the sample can leverage CREST resources in the groovy scripts, to create CREST requests.

As noted on the [Microsoft Azure AD web site](#), Azure is Microsoft's cloud platform for building, deploying, and managing applications and services.

This section assumes that you have a working knowledge of the Microsoft Azure AD platform, and can adjust and modify the configuration as needed to get the information needed to configure a Microsoft Azure AD connector.

You will also need to set up OpenIDM as a Microsoft Azure AD website and as a web application. When you do, retain the values of the URI of the website, `redirectURI`, the `APP ID URI`, which corresponds to the `resourceURI`, the `client ID`, and the `client Secret`.

The Groovy Connector Toolkit is bundled with OpenIDM 3.1, in the following JAR file: `openidm/connectors/groovy-connector-1.4.1.0.jar`.

The connector configuration file for this sample (`samples/scriptedazure/conf/provisioner.openidm-scriptedazure.json`) indicates the Scripted Azure AD implementation of the Groovy Connector Toolkit as follows:

```
{
  "name": "scriptedazure",
  "connectorRef": {
    "connectorHostRef": "#LOCAL",
    "connectorName": "org.forgerock.openicf.connectors.groovy.ScriptedConnector",
    "bundleName": "org.forgerock.openicf.connectors.groovy-connector",
    "bundleVersion": "[1.4.0.0,2.0.0.0)"
  },
  ...
}
```

The Groovy scripts required for the sample are located in the `samples/scriptedazure/tools` directory. You will need to 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.

3.7.4.1. Identifying Appropriate Information from Microsoft Azure AD

Microsoft Azure AD supports RESTful communication with the *Azure AD Graph API*. As suggested by the name, the API supports access to the Azure-based instance of Active Directory.

To get an OAuth 2 access token, you will need to add information to the Scripted Azure AD provisioner file, `provisioner.openicf-scriptedazure.json`. Substitute for the `__configureme__` entries shown.

```
...
"customConfiguration": "
  graphServiceUrl = 'https://graph.windows.net/__configureme__';
  oauth2 {
    authority = 'https://login.windows.net/__configureme__';
    redirectURI = '__configureme__';
    resourceURI = 'https://graph.windows.net'
  },
"customSensitiveConfiguration": "
  oauth2 {
    clientId = '__configureme__';
    clientSecret = '__configureme__';
    username = '__configureme__';
    password = '__configureme__'
  },...
```

The following list assumes that you have configured `contoso.com` as the Azure AD domain tenant. Substitute appropriately. Except for the first item, the following list of items are required for an OAuth 2 access token.

You should be able to get some of the items needed for the OAuth 2 token in the Azure AD configuration menu related to the OpenIDM web application.

- `graphServiceUrl`

For a domain tenant of `contoso.com`, set this to `'https://graph.windows.net/contoso.onmicrosoft.com`

- `authority`

Set to `'https://login.windows.net/contoso.onmicrosoft.com'`

- `redirectURI`

Set to `'https://localhost:8443/admin/index.html#azureCallback'`

- `resourceURI`

Keep the following Graph API setting: `'https://graph.windows.net'`

- `clientId`

Set to the Client ID shown in the Configure menu of the Azure AD web application.

- `clientSecret`

Set to the Client Secret shown when you added or configured the web application. In Azure AD, the Client Secret is the `key` associated with the Client ID.

- `username`

Set to the administrative username for Azure AD.

- `password`

Set to the administrative password for Azure AD.

3.7.4.2. Running the Sample

This section illustrates basic CRUD operations on users and groups using the scripted Azure AD connector. Note that the power of the Groovy connector is in the associated Groovy scripts, and their application in your specific deployment. The scripts provided with this sample are specific to the sample and customization of the scripts is required.

1. Start OpenIDM with the configuration for the scripted Azure AD sample.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/scriptedazure/
```

2. Check the connector configuration. Obtain its status over REST.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/system/scriptedazure?_action=test"

{
  "ok": true,
  "connectorRef": {
    "bundleVersion": "[1.4.0.0,2.0.0.0)",
    "bundleName": "org.forgerock.openicf.connectors.groovy-connector",
```

```

    "connectorName": "org.forgerock.openicf.connectors.groovy.ScriptedConnector"
  },
  "objectTypes": [
    "User",
    "Device",
    "Contact",
    "DeviceConfiguration",
    "ServicePrincipal",
    "Group",
    "ExtensionProperty",
    "OAuth2PermissionGrant",
    "DirectoryRole",
    "DirectoryObject",
    "DirectoryRoleTemplate",
    "AppRoleAssignment",
    "DirectoryLinkChange",
    "Application",
    "_ALL_",
    "SubscribedSku",
    "TenantDetail"
  ],
  "config": "config/provisioner.openicf/scriptedazure",
  "enabled": true,
  "name": "scriptedazure"
}

```

3. Create a group entry on the Azure AD server.

```

$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "_id": "fake placeholder value",
  "mailEnabled": false,
  "displayName": "Azure AD Test Group",
  "mailNickname": "Test_Group",
  "securityEnabled": "true",
  "description": "Azure AD Test Group"
}' \
"https://localhost:8443/openidm/system/scriptedazure/group?action=create"

{
  "mailEnabled": false,
  "__NAME__": "ab347420f-4e8e-f045-a437b62c0f3-636e",
  "mailNickname": "Test_Group",
  "displayName": "Azure AD Test Group",
  "mail": null,
  "onPremisesSecurityIdentifier": null,
  "lastDirSyncTime": null,
  "dirSyncEnabled": null,
  "description": "Azure AD Test Group",
  "proxyAddresses": [
  ],

```

```
"securityEnabled":true,
"provisioningErrors":[
],
"_id":"ab347420f-4e8e-f045-a437b62c0f3-636e"
}
```

4. Create a user entry on the Azure AD server.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "_id": "fake placeholder value",
  "accountEnabled": true,
  "displayName": "Alex Wu",
  "mailNickname": "AlexW",
  "passwordProfile": { "password" : "Test1234", "forceChangePasswordNextLogin": false },
  "userPrincipalName": "Alex@contoso.onmicrosoft.com"
}' \
"https://localhost:8443/openidm/system/scriptedazure/User?_action=create"

{
  "userType":"Member",
  "assignedLicenses":[
  ],
  "passwordPolicies":null,
  "immutableId":null,
  "surname":null,
  "provisionedPlans":[
  ],
  "state":null,
  "accountEnabled":true,
  "telephoneNumber":null,
  "physicalDeliveryOfficeName":null,
  "mail":null,
  "__NAME__":"744aa30f-55e0-4559-b956-985b77d80814",
  "proxyAddresses":[
  ],
  "otherMails":[
  ],
  "passwordProfile":null,
  "assignedPlans":[
  ],
  "facsimileTelephoneNumber":null,
  "displayName":"Alex Wu",
  "onPremisesSecurityIdentifier":null,
  "userPrincipalName":"Alex@contoso.onmicrosoft.com",
  "mobile":null,
  "dirSyncEnabled":null,
```



```

"jobTitle":null,
"givenName":null,
"department":null,
"usageLocation":null,
"preferredLanguage":null,
"sipProxyAddress":null,
"country":null,
"mailNickname":"AlexW",
"postalCode":null,
"lastDirSyncTime":null,
"city":null,
"provisioningErrors":[
],
"streetAddress":null,
"_id":"744aa30f-55e0-4559-b956-985b77d80814"
}

```

- Update Alex Wu's entry, by modifying his telephone number.

```

$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "telephoneNumber" : "555-1234"
}' \
"https://localhost:8443/openidm/system/scriptedazure/User/744aa30f-55e0-4559-b956-985b77d80814"
...
"telephoneNumber" : "555-1234"
...

```

- Confirm the result. Get the data for Alex Wu, by id.

```

$ curl \
--cacert self-signed.crt \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/scriptedazure/User/744aa30f-55e0-4559-b956-985b77d80814"

```

- Get information on the test group, also by id.

```

$ curl \
--cacert self-signed.crt \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/scriptedazure/Group/ab347420f-4e8e-f045-a437b62c0f3-636e"

```

- Delete the entry for user Alex Wu.

```
$ curl \
--cacert self-signed.crt \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request DELETE \
"https://localhost:8443/openidm/system/scriptedazure/User/744aa30f-55e0-4559-b956-985b77d80814"

{
  "_id": "744aa30f-55e0-4559-b956-985b77d80814"
}
```

9. Delete the group created earlier.

```
$ curl \
--cacert self-signed.crt \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request DELETE \
"https://localhost:8443/openidm/system/scriptedazure/Group/ab347420f-4e8e-f045-a437b62c0f3-636e"

{
  "_id": "ab347420f-4e8e-f045-a437b62c0f3-636e"
}
```

3.8. Sample - Using the PowerShell Connector Toolkit to Create Scripted Connectors

OpenICF 1.4 introduces a generic PowerShell Connector Toolkit that 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, with a subscription, from [ForgeRock Backstage](#).

This sample provides a number of PowerShell scripts that enable you to perform basic CRUD (create, read, update, delete) operations on an Active Directory server. The samples use the MS Active Directory PowerShell module. For more information on this module, see the corresponding Microsoft documentation.

The sample assumes that OpenIDM is running on the localhost, and that Active Directory runs on a remote server. The remote server also runs the OpenICF .NET connector server.

3.8.1. Setting Up the PowerShell Sample

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 Procedure 11.1, "Installing the .NET Connector Server" in the *Integrator's Guide*.

2. Configure OpenIDM to connect to the .NET connector server.

To do so, copy the remote connector provisioner file from the `openidm/samples/provisioners` directory to the `openidm/conf` directory, and edit the file according to your configuration.

```
$ cd /path/to/openidm
$ cp samples/provisioners/provisioner.openicf.connectorinfoprovider.json conf/
```

For instructions on editing this file, see Procedure 11.4, "Configuring OpenIDM to Connect to the .NET Connector Server" in the *Integrator's Guide*.

3. Download the PowerShell Connector Toolkit archive (`mspowershell-connector-1.4.1.0.zip`) from ForgeRock Backstage.

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 from the `tools` folder in this sample directory, to the machine on which the connector server is installed.

```
C:\>dir openidm\powershell2AD\tools
Directory of C:\openidm\powershell2AD\tools

11/27/2014  12:53 PM  <DIR>          .
11/27/2014  12:53 PM  <DIR>          ..
11/26/2014  02:32 PM                2,813 ADAAuthenticate.ps1
11/26/2014  02:32 PM                8,546 ADCreate.ps1
11/26/2014  02:32 PM                2,530 ADDelete.ps1
11/26/2014  02:32 PM                2,617 ADResolveUsername.ps1
11/26/2014  02:32 PM                8,644 ADSchema.ps1
11/26/2014  02:32 PM                3,458 ADSearch.ps1
11/26/2014  02:32 PM                4,585 ADSync.ps1
11/26/2014  02:32 PM                2,075 ADTest.ps1
11/26/2014  02:32 PM                8,488 ADUpdate.ps1
          9 File(s)          43,756 bytes
          2 Dir(s)  23,696,863,232 bytes free

C:\>
```

5. Edit the PowerShell search and sync scripts (`ADSearch.ps1`) and (`ADSync.ps1`) to specify the base suffix for your Active Directory Domain Controller.

Specifically, change the value of the `$searchBase` parameter in these files to match the suffix in your environment. The default suffix is:

```
$searchBase = 'CN=Users,DC=example,DC=com'
```

6. Copy the sample connector configuration for the PowerShell connector from the samples directory to your `conf` directory.

```
$ cd /path/to/openidm  
$ cp samples/provisioners/provisioner.openicf-adpowershell.json conf/
```

The sample connector configuration assumes that the scripts are located in `C:/openidm/samples/powershell2AD/tools/`. If you copied your scripts to a different location, adjust your connector configuration file accordingly.

Note that the OpenICF framework requires the path to use forward slash characters and not the backslash characters that you would expect in a Windows path.

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",
```

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.

3.8.2. Testing the PowerShell Sample

Because you have copied all of the required configuration files into the default OpenIDM project, you can start OpenIDM with the default configuration (that is, without the `-p` option).

```
$ cd /path/to/openidm  
$ ./startup.sh
```

When OpenIDM has started, you can 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 OpenIDM is able to connect to the .NET connector server with the following request.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/system?_action=test"
[
  {
    "ok": true,
    "connectorRef": {
      "bundleVersion": "1.4.1.0",
      "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 \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/adpowershell/account?_queryId=query-all-ids"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 1257,
  "result": [
    {
      "_id": "7c41496a-9898-4074-a537-bed696b6be92",
      "distinguishedName": "CN=Administrator,CN=Users,DC=example,DC=com"
    },
    {
      "_id": "f2e08a5c-473f-4798-a2d5-d5cc27c862a9",
      "distinguishedName": "CN=Guest,CN=Users,DC=example,DC=com"
    },
    {
      "_id": "99de98a3-c125-48dd-a7c2-e21f1488ab06",
      "distinguishedName": "CN=Ben Travis,CN=Users,DC=example,DC=com"
    },
    {
      "_id": "0f7394cc-c66a-404f-ad6d-38dbb4b6526d",
      "distinguishedName": "CN=Barbara Jensen,CN=Users,DC=example,DC=com"
    },
    {
      "_id": "3e6fa858-ed3a-4b58-9325-1fca144eb7c7",
      "distinguishedName": "CN=John Doe,CN=Users,DC=example,DC=com"
    }
  ]
}
```

```

    },
    {
      "_id": "6feef4a0-b121-43dc-be68-a96703a49aba",
      "distinguishedName": "CN=Steven Carter,CN=Users,DC=example,DC=com"
    }
  ,
  ...

```

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 \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/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 whether 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 \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/system/adpowershell/account?_action=authenticate&username=Steven+Carter&password=Passw0rd"
{
  "_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 Section 7.3.4, "Constructing Queries" in the *Integrator's Guide*.

The following query returns the record for the guest user.

```

$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/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
  }
}
}

```

6. Test whether you are able to 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 \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--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",
  "__ENABLE__" : true,
  "__PASSWORD__" : "Passw0rd",
  "telephoneNumber" : "0052-611-091"
}'

```



```

} \
"https://localhost:8443/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 whether you are able to update a user object on the Active Directory server by sending a PUT request with the complete object, and 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 \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "distinguishedName" : "CN=Jane Doe,CN=Users,DC=example,DC=com",
  "sn" : "Doe",
  "cn" : "Jane Doe",
  "sAMAccountName" : "sample",
  "userPrincipalName" : "janedoe@example.com",
  "ENABLE" : true,
  "PASSWORD" : "Passw0rd",
  "telephoneNumber" : "0052-611-091",
  "employeeNumber": "567893"
}' \
"https://localhost:8443/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 \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request DELETE \
"https://localhost:8443/openidm/system/adpowershell/account/42725210-8dce-4fdf-b0e0-393cf0377fdf"

```

The response includes the complete user object that was deleted.

You can you attempt to query the user object to confirm that it has been deleted.

```

$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/adpowershell/account/42725210-8dce-4fdf-b0e0-393cf0377fdf"
{
  "message": "",
  "reason": "Not Found",
  "code": 404
}

```

3.9. Sample 4 - CSV to XML File

Sample 4 works with two databases, a comma-separated value file and an XML file. There is no need to include any external resources.

A correlation query is used to relate the records in these two files.

3.9.1. Install the Sample

No external configuration is required for this sample. Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM", then start up OpenIDM with the configuration of sample 4.

```

$ cd /path/to/openidm
$ ./startup.sh -p samples/sample4

```

3.9.2. Check and then Run the Sample

The `sample4/data/hr.csv` file contains two example users. The first line of the file sets the attribute names.

Review the current contents of the database in the `sample4/data/xmlConnectorData.xml` file. For comparison purposes, make a copy of the file in a temporary directory with a command like:

```
$ cp /path/to/openidm/samples/sample4/data/xmlConnectorData.xml /tmp/
```

The reconciliation command run here uses the information from the `hr.csv` file to update the database in the `sample4/data/xmlConnectorData.xml` file.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/recon?_action=recon&mapping=csv_xmlfile&waitForCompletion=true"
```

Check the results of reconciliation. Review the updated contents of the `sample4/data/xmlConnectorData.xml` file.

If you want to experiment further, try changing the data in the `hr.csv` file. Run the noted reconciliation command again.

These users will not be visible from the OpenIDM UI, since they are mapped directly between the XML and CSV files. The internal OpenIDM repository is not updated in this sample.

3.10. Sample 5 - Synchronization of Two Resources

Sample 5 demonstrates the flow of data from one external resource to another. The resources are named `LDAP` and `AD` but in the sample, both directory-like resources are simulated with XML files.

You can optionally configure an outbound email service, if you want to receive emailed reconciliation summaries, as described in the following section.

3.10.1. 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 at the OpenIDM console when you run a reconciliation operation:

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

To configure an email summary, follow these steps:

1. Copy the template `external.email.json` file from the `/path/to/openidm/samples/misc` directory:

```
$ cd /path/to/openidm
$ cp samples/misc/external.email.json samples/sample5/conf
```

2. Edit the `external.email.json` file for outbound email, as described in Chapter 20, "Sending Email" in the *Integrator's Guide*.
3. Edit the `reconStats.js` file from the `/path/to/openidm/samples/sample5/script` directory. Near the start of the file, configure the OpenIDM email service to send statistics to the email addresses of your choice:

```
var params = {
  //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,
```

3.10.2. Install the Sample

No external configuration is required for this sample. Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM", then start OpenIDM with the configuration of sample 5.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/sample5
```

The XML files that simulate the resources are located in the `openidm/samples/sample5/data/` folder. When you start OpenIDM with the sample 5 configuration, OpenIDM creates the `xml_AD_Data.xml` file, which does not contain users until you run reconciliation.

3.10.3. Run the Sample

Run a reconciliation operation, to synchronize the contents of the simulated LDAP resource to the OpenIDM repository.

```
$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --header "Content-Type: application/json" \
  --request POST \
  "https://localhost:8443/openidm/recon?
_action=recon&mapping=systemLdapAccounts_managedUser&waitForCompletion=true"
```

This command creates a user in the repository. It is not necessary to run a second reconciliation operation to synchronize the AD resource. Automatic synchronization propagates any change made to managed users in the OpenIDM repository to the simulated AD resource.

Review the contents of `xml_AD_Data.xml`. It should now contain information for the same user that was present in the startup version of the `xml_LDAP_Data.xml` file.

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

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/ad/account?_queryId=query-all-ids"

{
  "result" : [ {
    "name" : "DDOE1",
    "_UID_" : "8dad9df3-820d-41ea-a3ab-a80c241bbc98",
    "_id" : "8dad9df3-820d-41ea-a3ab-a80c241bbc98"
  } ],
  "resultCount" : 1,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : -1
}
```

You can use the `_id` of the user to read the user information from the AD resource, for example:

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/ad/account/8dad9df3-820d-41ea-a3ab-a80c241bbc98"

{
  "email" : [ "mail@example.com" ],
  "name" : "DDOE1",
  "_UID_" : "8dad9df3-820d-41ea-a3ab-a80c241bbc98",
  "firstname" : "Darth",
  "lastname" : "Doe",
  "_id" : "8dad9df3-820d-41ea-a3ab-a80c241bbc98"
}
```

To verify that the sample is working, repeat the process. Set up a second user in the `xml_LDAP_Data.xml` file. An example of how that file might appear with a second user (`GD0E1`) is shown here:

```
<?xml version="1.0" encoding="UTF-8"?>
<icf:OpenICFContainer
  xmlns:icf="http://openidm.forgerock.com/xml/ns/public/resource/openicf/resource-schema-1.xsd"
  xmlns:ri="http://openidm.forgerock.com/xml/ns/public/resource/instances/resource-schema-extension"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://openidm.forgerock.com/xml/ns/public/resource/instances/resource-schema-
extension
/path/to/openidm/samples/sample5/data/resource-schema-extension.xsd
http://openidm.forgerock.com/xml/ns/public/resource/openicf/resource-schema-1.xsd
/path/to/openidm/samples/sample5/data/resource-schema-1.xsd">
  <ri:__ACCOUNT__>
    <icf:__UID__>1</icf:__UID__>
    <icf:__PASSWORD__>TestPassw0rd2</icf:__PASSWORD__>
    <ri:firstname>Darth</ri:firstname>
    <icf:__DESCRIPTION__>Created By XML1</icf:__DESCRIPTION__>
    <icf:__NAME__>DDOE1</icf:__NAME__>
    <ri:email>mail1@example.com</ri:email>
    <ri:lastname>Doe</ri:lastname>
  </ri:__ACCOUNT__>
  <ri:__ACCOUNT__>
    <icf:__UID__>2</icf:__UID__>
    <icf:__PASSWORD__>TestPassw0rd2</icf:__PASSWORD__>
    <ri:firstname>Garth</ri:firstname>
    <icf:__DESCRIPTION__>Created By XML1</icf:__DESCRIPTION__>
    <icf:__NAME__>GDOE1</icf:__NAME__>
    <ri:email>mail2@example.com</ri:email>
    <ri:lastname>Doe</ri:lastname>
  </ri:__ACCOUNT__>
</icf:OpenICFContainer>
```

Rerun the reconciliation and query REST commands shown previously. The reconciliation operation creates the new user from the simulated LDAP resource in the OpenIDM repository. An implicit synchronization operation then creates that user in the AD resource.

3.11. Sample 5b - Failure Compensation With Multiple Resources

The compensated synchronization mechanism depicted in this sample can help manage the risks associated with synchronizing data across multiple resources.

Typically, when a managed/user object is changed, implicit synchronization replays that change to all configured external resources. If synchronization fails for one target resource (for example, due to a policy validation failure on the target, or the target being unavailable), the synchronization operation stops at that point. The effect is that a record might be changed in the repository, and in the targets on which synchronization was successful, but not on the failed target, or any targets that would have been synchronized after the failure. This situation can result in disparate data sets across resources. While a reconciliation operation would eventually bring all targets back in sync, reconciliation can be an expensive operation with large data sets.

The compensated synchronization mechanism ensures that either all resources are synchronized successfully, or that the original change is rolled back. This mechanism uses an `onSync` script hook

configured with a `compensate.js` script that can be used to "revert" the partial change to managed/user and to the corresponding external resources.

Sample 5b is similar to sample 5 in that it simulates two external resources with XML files (located in the `/path/to/samples/sample5b/data` directory). The `xml_LDAP_Data.xml` file simulates an LDAP data source. OpenIDM creates the `xml_AD_Data.xml` file when you start OpenIDM with the sample. Sample 5b adds the `onSync` script hook to the process, configured in the `sample5b/conf/managed.json` file.

The following excerpt of the `managed.json` file shows the `onSync` hook, which calls the `compensate.js` script, provided in the `/path/to/openidm/bin/defaults/script` directory.

```
...
},
"onSync" : {
  "type" : "text/javascript",
  "file" : "compensate.js"
},
```

You can use the `onSync` script hook to ensure that changes made in the repository are synchronized to all external resources, or that no changes are made. For more information about how implicit synchronization uses the `onSync` script hook, see Section 12.12, "Configuring Synchronization Failure Compensation" in the *Integrator's Guide*.

You can optionally configure an outbound email service for this sample, if you want to receive emailed reconciliation summaries. The email service configuration is identical to that of sample 5 (Section 3.10.1, "Configure Email for the Sample").

3.11.1. Install the Sample

No external configuration is required for this sample. Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM", then start OpenIDM with the configuration of sample 5b.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/sample5b
```

The XML files that simulate an external LDAP and AD resource are now located in the `openidm/samples/sample5b/data/` directory. The simulated AD data store file, `xml_AD_Data.xml`, does not contain users until you run reconciliation.

Run the sample in exactly the same way that you did for Sample 5, following the steps in Section 3.10.3, "Run the Sample".

3.11.2. Demonstrate onSync

To demonstrate integration of the samples with the OpenIDM UI, this sample uses the UI to view and make changes to user objects in the repository. However, you can also use the REST interface to make these changes, as shown in the previous section.

Log into the OpenIDM UI as the administrative user. On a local system, navigate to <https://localhost:8443/openidmui>. The default administrative account and password are both `openidm-admin`.

Make a change to the data of an existing user (`DDOE1`). As a result of the implicit synchronization from the managed object repository, that change is reflected almost immediately on the external resources. For sample 5b, you should see the changes in both XML files in the `/path/to/openidm/sample5b/data` directory. Alternatively, you can query the external resources over REST, as described previously.

The synchronization is successful, across all configured external resources, so the updated user record can be seen in both the `xml_LDAP_Data.xml` and `xml_AD_Data.xml` files.

The next step is to simulate a problem connecting to the LDAP resource. One way to do so on the local system is to rename the LDAP data file so that it is unreadable. On a Linux system, the following command, as an administrative user, would serve that purpose:

```
$ cd /path/to/openidm/samples/sample5b/data
$ sudo mv xml_LDAP_Data.xml xml_LDAP_Data.xml.bak
```

In the UI, now try another update to user `DDOE1`. With the modified filename of the simulated LDAP resource, implicit synchronization cannot write to this resource. An error similar to the following is displayed in the log file, `openidm0.log.0`:

```
Data file does not exist:
/path/to/openidm/samples/sample5b/data/xml_LDAP_Data.xml
```

Although the AD resource is available, implicit synchronization will not reach this resource, because the mapping is specified *after* the managed/user to LDAP mapping in the `sync.json` file.

When the implicit synchronization operation fails for the LDAP resource, the `onSync` hook invokes the `compensate.js` script. This script attempts to revert the original change by performing another update to `DDOE1` in the repository (managed/user). This change, in turn, triggers another automatic synchronization to the AD and LDAP resources.

Because the LDAP resource is still unreadable, the synchronization to LDAP fails again, which triggers the `compensate.js` script again. This time, however, the script recognizes that the change was originally called as a result of a compensation and aborts.

The original synchronization error from the first update is thrown from the script and the UI should display that error. If you refresh the UI, and view that user entry again, you will notice that the change to the entry has been reverted.

Note that if you change the name of the AD resource file (to make it unavailable), a change to a managed/user entry will be synchronized successfully with the LDAP resource (because that mapping appears first in `sync.json`). The synchronization will fail for the AD resource. In this case, the change will be reverted on both the managed/user entry, and the LDAP resource.

3.12. Sample 6 - LiveSync With an AD Server

Sample 6 resembles sample 5, but demonstrates LiveSync from an external resource. Sample 6 includes configuration files for two scenarios, depending on whether you have a live Active Directory (AD) service, or whether you need to simulate an AD service with an OpenDJ server. Each scenario is associated with a file in the `/path/to/sample6/alternatives` directory. The file you select should be copied to the `/path/to/sample6/conf` directory.

Active AD Deployment

If you have an actual AD deployment available, copy the `provisioner.openicf-realad.json` file to the `conf/` subdirectory. You can then configure synchronization between an OpenDJ Directory Server and an active AD deployment.

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

Simulated AD Deployment

If you need to simulate an AD deployment, copy the `provisioner.openicf-fakead.json` file to the `conf/` subdirectory. You can then configure synchronization between an OpenDJ Directory server and a simulated AD server.

This sample simulates an AD server on the same instance of OpenDJ, using a different base DN.

The options shown in the associated configuration files can be easily modified to work with any standard LDAP server.

3.12.1. Sample 6 with an Active AD Deployment

If you have an existing, active instance of AD, set up OpenDJ, as described in the *OpenDJ Installation Guide*.

During installation, populate OpenDJ with the data in the `Example.ldif` file, available in the `/path/to/sample6/data` directory.

The actions run in this sample should not change any data on the AD server.

3.12.2. Sample 6 with a Simulated AD Deployment

In this sample, an AD deployment is simulated with a different baseDN (`dc=fakead,dc=com`) on the same OpenDJ server instance. You can also simulate the AD server with a separate OpenDJ 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 `AD.ldif`. The data for the OpenDJ instance is contained in the file `Example.ldif`.

3.12.3. External Configuration

3.12.3.1. Prepare OpenDJ For LiveSync

This sample assumes a replicated OpenDJ server on the localhost system. When configured, OpenDJ replication includes an External Change Log (ECL), required to support LiveSync. LiveSync detects changes in OpenDJ by reading the ECL.

Follow these steps to install and configure an OpenDJ instance.

1. Download and extract the OpenDJ zip archive from the [OpenDJ Project page](#).
2. Install OpenDJ using the command-line setup, as follows:

```
$ cd /path/to/opendj
$ ./setup --cli
\
--hostname localhost
\
--ldapPort 1389
\
--rootUserDN "cn=Directory Manager"
\
--rootUserPassword password
\
--adminConnectorPort 4444
\
--addBaseEntry
\
--baseDN dc=com
\
--acceptLicense
\
--no-prompt
...
Configuring Directory Server ..... Done.
Creating Base Entry dc=com ..... Done.
Starting Directory Server ..... Done
.
...
```

The sample assumes the following configuration:

- The OpenDJ server is installed on the localhost.
- The server listens for LDAP connections on port 1389.
- The administration connector port is 4444.
- The root user DN is `cn=Directory Manager`.
- The root user password is `password`.

3. Import the LDIF data file required for the sample.

```
$ cd /path/to/opensj/bin
$ ./ldapmodify \
  --bindDN "cn=Directory Manager" \
  --bindPassword password \
  --hostname localhost \
  --port 1389 \
  --filename /path/to/openidm/samples/sample6/data/Example.ldif
```

4. Configure the OpenDJ server for replication.

To enable liveSync, this server must be configured for replication, even if it does not actually participate in a replication topology. The following commands configure the server for replication.

```
$ ./dsconfig create-replication-server \
  --hostname localhost \
  --port 4444 \
  --bindDN "cn=Directory Manager" \
  --bindPassword password \
  --provider-name "Multimaster Synchronization" \
  --set replication-port:8989 \
  --set replication-server-id:2 \
  --type generic \
  --trustAll \
  --no-prompt

$ ./dsconfig create-replication-domain \
  --hostname localhost \
  --port 4444 \
  --bindDN "cn=Directory Manager" \
  --bindPassword password \
  --provider-name "Multimaster Synchronization" \
  --domain-name fakead_com \
  --set base-dn:dc=fakead,dc=com \
  --set replication-server:localhost:8989 \
  --set server-id:3 \
  --type generic \
  --trustAll \
  --no-prompt
```

Once OpenDJ is configured, you can proceed with either an active or simulated AD deployment.

3.12.3.2. External Configuration for an Active AD Deployment

To configure an active AD deployment for sample 6, open the `provisioner.openicf-realad.json` file in a text editor. Update it as needed. At minimum, you should check and if needed update the following parameters in that file, as shown in the following table:

Table 3.1. Key Parameters for an Active AD Configuration

Option	Description
host	The hostname/IP address of the AD server
port	The LDAP port; the default is 389.
ssl	By default, SSL is not used.
principal	The full DN of the account to bind with, such as "CN=Administrator,CN=Users,DC=example,DC=com"
credentials	If a password is used, replace null with that password. When OpenIDM starts, it encrypts that password in the <code>provisioner.openicf-realad.conf</code> file.
baseContexts	The DN's for account containers, such as ["CN=Users,DC=Example,DC=com"]
baseContextsToSynchronize	Set to the same value as <code>baseContexts</code>
accountSearchFilter	Default searches for active user (not computer) accounts
accountSynchronizationFilter	Default synchronizes with active user (not computer) accounts

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

3.12.3.3. External Configuration for a Simulated AD Deployment

Not everyone has a testable instance of AD readily available. For such administrators, you can use the `AD.ldif` file from the `data/` subdirectory to simulate an AD deployment.

If you have not already done so, copy the `provisioner.openicf-fakead.json` file to the `conf` subdirectory.

As previously mentioned, you can use a separate OpenDJ instance to simulate the AD server. However, the following instructions assume that the simulated AD server runs on the same OpenDJ instance.

Open the `provisioner.openicf-fakead.json` file and note the following:

- OpenDJ directory server uses port 1389 by default for users who cannot use privileged ports, so this is the port that is specified in the provisioner file. Adjust the port if your OpenDJ server is listening on a different port.
- The simulated AD server uses the base DN `dc=fakead,dc=com`.

To load the data for the simulated AD instance, launch the OpenDJ control panel, add the simulated AD baseDN (`dc=fakead,dc=com`), and then import the `/path/to/sample6/data/AD.ldif` file. When you import the `AD.ldif` file, select "Append to Existing Data", not "Overwrite Existing Data". Otherwise, the data in the `dc=example,dc=com` baseDN will be overwritten.

3.12.4. Start OpenIDM with Sample 6

Now that OpenDJ and a real or simulated AD database is configured, prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM". You can then start OpenIDM with the configuration for sample 6.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/sample6
```

3.12.5. Run the Sample

The following sections show how to run the sample with command-based reconciliation with a REST call, and to configure scheduled reconciliation with LiveSync.

3.12.5.1. Using Reconciliation

Now that OpenIDM is in operation, review the entries in the OpenDJ data store. 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 Section 3.12.3.3, "External Configuration for a Simulated AD Deployment", compare the entries for `uid=jdoe` as shown in the `AD.ldif` and `Example.ldif` files. Note the different values of `givenName` for `uid=jdoe`.

Run reconciliation over the REST interface. If you have followed the instructions for the simulated AD data store, the following command takes the information for user `jdoe` imported from the `AD.ldif` file, with a `givenName` of Johnny, and synchronizes that information to the LDAP database, overwriting the `givenName` of John for that same user `jdoe`.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/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 deployment with OpenIDM repository (managed/user). That information is then automatically synchronized to the OpenDJ server, as described in Section 12.13, "Synchronization Situations and Actions" in the *Integrator's Guide*.

After reconciliation, list all users in the OpenDJ server data store.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/ldap/account?_queryId=query-all-ids"
```

The result should resemble the following JSON object.

```
{
  "result": [ {
    "dn" : "uid=jdoe,ou=People,dc=example,dc=com",
    "_id" : "uid=jdoe,ou=People,dc=example,dc=com"
  }, {
    "dn" : "uid=bjensen,ou=People,dc=example,dc=com",
    "id" : "uid=bjensen,ou=People,dc=example,dc=com"
  } ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1,
}
```

You see only two entries, as the `uid=jdoe` entry from `dc=fakead,dc=com` overwrites the original LDAP entry for `uid=jdoe` in the reconciled LDAP data store.

To read the user object in the OpenDJ server, run the `ldapsearch` command. The following example returns the entry for user `uid=jdoe`:

```
$ ./ldapsearch \
--port 1389 \
--baseDN dc=example,dc=com \
"(uid=jdoe)"
```

3.12.5.2. Using LiveSync

You can start reconciliation by using a scheduled configuration or by using the REST interface directly. However, to use LiveSync, you must configure a schedule. When LiveSync is active, the default schedule in the `schedule-activeSynchroniser_systemAdAccount.json` configuration file runs LiveSync every 15 seconds.

LiveSync pushes changes made in the AD data store to the OpenIDM repository, automatically.

LiveSync is disabled by default. To activate LiveSync, change the value of the `"enabled"` property from `false` to `true`.

```
{
  "enabled" : false,
  "type" : "cron",
  "schedule" : "0/15 * * * * ?",
  "invokeService" : "provisioner",
  "invokeContext" : {
    "action" : "liveSync",
    "source" : "system/ad/account"
  },
  "invokeLogLevel" : "debug"
}
```

Procedure 3.9. Testing LiveSync

Now you can test LiveSync. This procedure assumes that you have configured OpenDJ using the parameters and commands described in this section.

1. Create an LDIF file with a new user entry (`uid=bsmith`) that will be added to the simulated AD data store.
2. The following is the contents of a sample `bsmith.ldif` file for demonstration purposes:

```
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: passw0rd
```

3. Navigate to the `/path/to/opendj/bin` directory.
4. Use the `ldapmodify` command to add the `bsmith.ldif` file to the directory.

```
$ ./ldapmodify
--port 1389
--defaultAdd
--bindDN "cn=Directory Manager"
--bindPassword password
--filename /path/to/bsmith.ldif
```

5. Now you can test synchronization by viewing the new user in the OpenIDM repository. The easiest way to do this, is through OpenIDM UI. You should be able to log into the UI with any of the accounts in the AD data store. For this example, log into the UI as user `bsmith`, with password `passw0rd`. The fact that you can log into the UI as this new user indicates that LiveSync has synchronized the user from the AD data store to the managed/user repository.

6. Implicit synchronization pushes this change out to the OpenDJ data store. To test this synchronization operation, search the OpenDJ baseDN for the new user entry.

```
$ ./ldapsearch \  
--port 1389 \  
--baseDN ou=people,dc=example,dc=com \  
"(uid=bsmith)"
```

3.13. Sample 7 - Scripting a SCIM-like Schema

Sample 7 demonstrates how you can use OpenIDM to expose user data with a SCIM-like schema. The sample uses the XML file connector to read in attributes from external accounts and construct a JSON object for users stored in the OpenIDM repository. For more information about SCIM schema, see *System for Cross-Domain Identity Management: Core Schema 1.1*.

3.13.1. Install the Sample

Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM", then start OpenIDM with the configuration for sample 7.

```
$ cd /path/to/openidm  
$ ./startup.sh -p samples/sample7
```

3.13.2. Running the Sample

Run a reconciliation to pull the user from `samples/sample7/data/xmlConnectorData.xml` into the OpenIDM internal repository.

```
$ curl \  
--cacert self-signed.crt \  
--header "X-OpenIDM-Username: openidm-admin" \  
--header "X-OpenIDM-Password: openidm-admin" \  
--header "Content-Type: application/json" \  
--request POST \  
"https://localhost:8443/openidm/recon?  
_action=recon&mapping=systemXmlfileAccounts_managedUser&waitForCompletion=true"
```

Reconciliation creates a user object in the repository. Retrieve the user from the repository.

```
$ curl \  
--cacert self-signed.crt \  
--header "X-OpenIDM-Username: openidm-admin" \  
--header "X-OpenIDM-Password: openidm-admin" \  
--request GET \  
"https://localhost:8443/openidm/managed/user/DDO0E1"
```

The user object has the following JSON representation.

```
{  
  "_id" : "DDO0E1",  
  "_rev" : "2",  
  "schemas" : ["urn:scim:schemas:core:1.0"],
```

```

"ims" : [ {
  "type" : "aim",
  "value" : "jonyOnAim"
}, {
  "type" : "skype",
  "value" : "skyperHiasl"
} ],
"locale" : null,
"phoneNumbers" : [ {
  "type" : "work",
  "value" : "1234567"
}, {
  "type" : "home",
  "value" : "1234568"
} ],
"emails" : [ {
  "type" : "work",
  "value" : "hallo@example.com",
  "primary" : true
}, {
  "type" : "home",
  "value" : "jdoe@forgerock.com"
} ],
"externalId" : "DDOE1",
"preferredLanguage" : "en_US",
"meta" : {
  "lastModified" : "Thu May 01 2014 12:57:10 GMT-0800 (GMT-08:00)",
  "created" : "Thu May 01 2014 12:56:27 GMT-0800 (GMT-08:00)"
},
"userType" : "permanent",
"photos" : [ {
  "type" : "photo",
  "value" : "https://photos.example.com/profilephoto/7293000000Ccne/F"
}, {
  "type" : "thumbnail",
  "value" : "https://photos.example.com/profilephoto/7293000000Ccne/T"
} ],
"title" : "Mr.Universe",
"timezone" : "America/Denver",
"profileUrl" : "https://login.example.com/DDOE1",
"nickName" : "Jonny",
"name" : {
  "familyName" : "Doe",
  "middleName" : "Hias",
  "formatted" : "Dr. John H Doe III",
  "givenName" : "John",
  "honorificSuffix" : "III",
  "honorificPrefix" : "Dr."
},
"userName" : "DDOE1",
"displayName" : "John Doe",
"addresses" : [ {
  "streetAddress" : "100 Universal City Plaza",
  "region" : "CA",
  "formatted" : "100 Universal City Plaza\nHollywood, CA 91608 USA",
  "postalCode" : "91608",
  "primary" : "true",
  "locality" : "Hollywood",
  "type" : "work",

```

```
"country" : "USA"
}, {
  "streetAddress" : "222 Universal City Plaza",
  "region" : "CA",
  "formatted" : "222 Universal City Plaza\nHollywood, CA 91622 USA",
  "postalCode" : "91622",
  "primary" : "false",
  "locality" : "Hollywood",
  "type" : "home",
  "country" : "USA"
} ],
"groups" : [ {
  "value" : "usemploys",
  "display" : "US Employees"
}, {
  "value" : "euemploys",
  "display" : "EU Employees"
} ]
}
```

The sample script file, `scim.js` in the `/path/to/samples/sample7/script` directory, transforms the user data from the resource into the JSON object layout required by SCIM schema.

3.14. Sample 8 - Logging in Scripts

OpenIDM provides a `logger` object with `debug()`, `error()`, `info()`, `trace()`, and `warn()` functions that you can use to log messages to the OpenIDM console from your scripts.

3.14.1. Install the Sample

Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM", then start OpenIDM with the configuration for sample 8.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/sample8
```

The `sync.json` file in the `/path/to/samples/sample8/conf` directory includes brief examples of log messages.

3.14.2. Running the Sample

Run reconciliation over the REST interface.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/recon?
_action=recon&mapping=systemXmlfileAccounts_managedUser&waitForCompletion=true"
```

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

Notice the log messages displayed on the OpenIDM (Felix) console. The following example omits timestamps and so forth to show only the message strings.

```
->
...Case no Source: the source object contains: = null [5235432-...
...Case emptySource: the source object contains: = {lastname=Carter, mobile...
...Case sourceDescription: the source object contains: = Created By XML1
...Case onCreate: the source object contains: = {lastname=Carter, mobile...
...Case result: the source object contains: = {SOURCE_IGNORED={count=0, ids=[]},...
```

3.15. Sample 9 - Asynchronous Reconciliation Using Workflows

Sample 9 demonstrates asynchronous reconciliation using workflows. Reconciliation generates an approval request for each ABSENT user. 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 the request is approved by an administrator, the absent users are created by an asynchronous reconciliation process.

Prepare a fresh installation of OpenIDM before trying this sample.

3.15.1. Install the Sample

Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM", then start OpenIDM with the configuration for sample 9.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/sample9
```

3.15.2. Running the Sample

1. Run reconciliation over the REST interface.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/recon?
_action=recon&mapping=systemXmlfileAccounts_managedUser&waitForCompletion=true"
```

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

Reconciliation starts an approval workflow for each ABSENT user. These approval workflows (named `managedUserApproval`) wait for the request to be approved by an administrator.

2. Query the invoked workflow task instances over REST.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/workflow/taskinstance?_queryId=query-all-ids"
```

In this case, the request returns two workflow results, each with a process ID (`_id`) as well as a process definition ID. You will use the value of the `_id` shortly.

```
{
  "result" : [ {
    "tenantId" : "",
    "createTime" : "2014-05-01T13:48:42.980-08:00",
    "executionId" : "101",
    "delegationStateString" : null,
    "processVariables" : { },
    "_id" : "123",
    "_processInstanceId" : "101",
    "description" : null,
    "priority" : 50,
    "name" : "Evaluate request",
    "dueDate" : null,
    "parentTaskId" : null,
    "processDefinitionId" : "managedUserApproval:1:3",
    "taskLocalVariables" : { },
    "suspensionState" : 1,
    "assignee" : "openidm-admin",
    "cachedElContext" : null,
    "queryVariables" : null,
    "activityInstanceVariables" : { },
    "deleted" : false,
    "suspended" : false,
    "_rev" : 1,
    "_revisionNext" : 2,
    "category" : null,
    "taskDefinitionKey" : "evaluateRequest",
    "owner" : null,
    "eventName" : null,
    "delegationState" : null
  }, {
```

```

"tenantId" : "",
"createTime" : "2014-05-01T13:48:42.980-08:00",
"executionId" : "102",
"delegationStateString" : null,
"processVariables" : { },
"_id" : "124",
"processInstanceId" : "102",
"description" : null,
"priority" : 50,
"name" : "Evaluate request",
"dueDate" : null,
"parentTaskId" : null,
"processDefinitionId" : "managedUserApproval:1:3",
"taskLocalVariables" : { },
"suspensionState" : 1,
"assignee" : "openidm-admin",
"cachedElContext" : null,
"queryVariables" : null,
"activityInstanceVariables" : { },
"deleted" : false,
"suspended" : false,
"_rev" : 1,
"revisionNext" : 2,
"category" : null,
"taskDefinitionKey" : "evaluateRequest",
"owner" : null,
"eventName" : null,
"delegationState" : null
} ],
"resultCount" : 2,
"pagedResultsCookie" : null,
"remainingPagedResults" : -1
}

```

3. Approve the requests over REST, by setting the `requestApproved` parameter for the specified task instance to `true`. Note the use of one of the values of `_id` in the REST call, in this case, `124`.

On UNIX:

```

$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{"requestApproved": "true"}' \
"https://localhost:8443/openidm/workflow/taskinstance/124?_action=complete"

```

On Windows:

```
$ curl ^
--cacert self-signed.crt ^
--header "X-OpenIDM-Username: openidm-admin" ^
--header "X-OpenIDM-Password: openidm-admin" ^
--header "Content-Type: application/json" ^
--request POST ^
--data "{\"requestApproved\": \"true\"}" ^
"https://localhost:8443/openidm/workflow/taskinstance/124?_action=complete"
```

A successful call returns the following:

```
{"Task action performed":"complete"}
```

4. Once the request has been approved, an asynchronous reconciliation operation runs, which creates the users whose accounts were approved in the previous step.

List the users that were created by the asynchronous reconciliation.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/managed/user?_queryId=query-all-ids"
```

One user is returned.

```
{
  "result": [ {
    "_rev": "0",
    "_id": "1"
  } ],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1,
}
```

3.16. Sample - Configuring Authentication Management With OpenAM

This sample demonstrates how you can protect a deployment of OpenIDM with ForgeRock's OpenAM Access Management product.

In this sample, you will use OpenIDM together with OpenAM and ForgeRock's OpenDJ Directory Services product.

When configured together, you can demonstrate one use case that maximizes Identity Relationship Management (IRM) functionality. OpenIDM can maintain user data in OpenDJ data stores. Once protected by OpenAM, you can configure rules for authorized access.

This sample configures OpenIDM on one system, with a URL of `openidm.example.com`. It configures OpenAM and OpenDJ on a separate second system, with a URL of `openam.example.com`.

Note

This sample assumes that you have deployed OpenAM 12.

3.16.1. Prepare the Sample

Before installing OpenIDM, you need to prepare your systems. To set up this sample, you need to take the following basic steps:

- Install OpenDJ, using a base DN of `dc=example,dc=com`. Be sure to import the `Example.ldif` file available in the `samples/openam/data` directory. This example assumes that you install OpenDJ on the same system as OpenIDM, with a FQDN of `openidm.example.com`.

You can import the appropriate `Example.ldif` file with the following command.

```
$ cd /path/to/opendj/bin
$ ./ldapmodify \
--bindDN "cn=Directory Manager" \
--bindPassword password \
--hostname localhost \
--port 1389 \
--filename /path/to/openidm/samples/openam/data/Example.ldif
```

If you set up OpenDJ on a secure port with a self-signed certificate, you should import that certificate into the OpenIDM datastore. For more information, see Section 16.1, "Accessing the Security Management Service" in the *Integrator's Guide*.

- Install an OpenAM 12 server in an environment that OpenIDM can access with RESTful calls. For installation instructions, see the *OpenAM Installation Guide*. This example assumes that you have deployed the associated `.war` file in your selected Java EE container with the following name: `openam.war`.

Configure the OpenAM JVM on a secure port such as 8443.

When installing OpenAM, configure it to use the OpenDJ data store installed on `openam.example.com`.

As described in *To Custom Configure OpenAM*, make sure the trust store used by the JVM running OpenAM has the necessary certificates installed.

This sample assumes that you will configure OpenAM-based Single Sign-On (SSO) as described in the OpenAM Administration Guide. For this sample, configure `.example.com` as the sole cookie domain as described in the following section of the OpenAM Reference Guide: *System Configuration: Platform*.

Note

When you configure OpenAM for this sample, you do not need to install a Java EE agent. This sample uses the `OPENAM_SESSION` module to protect OpenIDM. That module is included in the `/path/to/openam/conf/authentication.json` file.

- Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM".
- As OpenAM works with FQDNs (and not IP addresses), you will need to set up either the noted FQDNs (`openam.example.com` and `openidm.example.com`) in an address database such as a DNS server or the static `hosts` configuration file of both systems.
- Do not start OpenIDM yet. You will need to configure it further.

3.16.2. Configuring OpenIDM for the OpenAM Sample

Before starting OpenIDM, you will need to change two configuration files, related to authentication and provisioning.

3.16.2.1. Configuring the authentication.json File for OpenAM

To configure OpenIDM authentication, open the `authentication.json` file. For this sample, you can find that file in the `samples/openam/conf` directory.

Under `"authModules"`, find the `"OPENAM_SESSION"` authentication module. The default version of the `authentication.json` file includes two entries which you must fill in:

```
"authModules" : [
  {
    "name" : "OPENAM_SESSION",
    "properties" : {
      "openamDeploymentUrl" : "",
      "groupRoleMapping" : {
        "openidm-admin" : [ ]
      }
    },
  },
]
```

Based on a standard `openidm-admin` user and the previously noted URL and `.war` file for your OpenAM deployment, you would change the code snippet to:

```
"authModules" : [
  {
    "name" : "OPENAM_SESSION",
    "properties" : {
      "openamDeploymentUrl" : "https://openam.example.com:8443/openam",
      "groupRoleMapping" : {
        "openidm-admin" : [
          "cn=idmAdmins,ou=Groups,dc=example,dc=com"
        ]
      }
    },
  },
]
```

Do remember to include `/openam` in the value of `"openamDeploymentUrl"`. After the Java EE container used for OpenAM starts, it unpacks a file such as `openam.war` so that you can access it on the `/openam` endpoint.

With the `"groupRoleMapping"`, you can add the Distinguished Names (DN) of groups to the list. This feature will then assign the noted role, such as `openidm-admin` to any user who is a member of one of these groups. For more information on how roles are assigned, see Section 15.6, "Roles and Authentication" in the *Integrator's Guide*.

The `"openamDeploymentUrl"` shown above assumes that you are using SSL. If you have a self-signed certificate, you will need to import that into the OpenIDM truststore file. For more information, see Section 16.1, "Accessing the Security Management Service" in the *Integrator's Guide*.

3.16.2.2. Configure Provisioning for the OpenAM Sample

This section describes how you can customize the `provisioner.openicf-ldap.json` file. The following edits parallel those described in Section 3.5, "Sample 2c - Synchronizing LDAP Group Membership".

Open the noted provisioner file, which you can find in the `samples/openam/conf` directory. In the initial version of this file, you should see something similar to the following code:

```
{
  "name" : "ldap",
  "configurationProperties" : {
    "host" : "localhost",
    "port" : 1389,
    "ssl" : false,
    "principal" : "cn=Directory Manager",
    "credentials" : "password",
    "baseContexts" : [
      "dc=example,dc=com"
    ],
    "baseContextsToSynchronize" : [
      "dc=example,dc=com"
    ],
  },
}
```

This snippet already matches the noted base context of `"dc=example,dc=com"` with a principal of `"cn=Directory Manager"`.

Make sure that the following settings are consistent with the way you have configured OpenDJ and OpenAM.

Change the `"localhost"` entry to the URL where OpenDJ is installed. In this case, that URL is `openidm.example.com`. Depending on whether you want to set up communications over a regular or secure LDAP port, you might change the `"port"` number to 389 or 636. The following excerpt illustrates the change to a secure connector configuration:

```
{
  "name" : "ldap",
  "configurationProperties" : {
    "host" : "openidm.example.com",
    "port" : 636,
    "ssl" : true,
    "principal" : "cn=Directory Manager",
    "credentials" : "password",
    "baseContexts" : [
      "dc=example,dc=com"
    ],
    "baseContextsToSynchronize" : [
      "dc=example,dc=com"
    ],
  },
}
```

Just remember, if you want to set up secure communications with OpenDJ, you should configure OpenDJ with that in mind. If you have a self-signed certificate for OpenDJ, you will also want to import that into the OpenIDM truststore, as described in Section 16.1, "Accessing the Security Management Service" in the *Integrator's Guide*.

3.16.3. Install the Sample

If you have not already done so, make sure OpenDJ is running on its assigned system, in this case, [openidm.example.com](#).

Finish preparing OpenIDM, with the procedure described in Section 3.1.2, "Preparing OpenIDM", then start OpenIDM with the configuration for sample openam.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/openam
```

If you have not already done so, make sure OpenAM is also running on its assigned system, in this case, [openam.example.com](#).

3.16.4. Running the Sample

With everything configured, now run reconciliation on the [openidm.example.com](#) system. The following command imports users from the OpenDJ database into the OpenIDM managed/user database:

```
$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --header "Content-Type: application/json" \
  --request POST \
  "https://localhost:8443/openidm/recon?
_action=recon&mapping=systemLdapAccounts_managedUser&waitForCompletion=true"
```

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

3.16.5. Make Sure the Sample Works as Intended

You should still be able to access the OpenIDM UI in the "normal" way. In this case, you would point a browser to <https://openidm.example.com:8443/openidmui>.

The difference is how OpenIDM uses a proxy service to authenticate with OpenAM's REST-based authentication service. This sample implements the proxy service as a custom endpoint in the `openamProxy.js` file, in the `samples/openam/script` directory.

The first time you log into the OpenIDM UI, you should see a login prompt configured with OpenAM. The default version of this prompt appears quite similar to the OpenIDM prompt. When you login through OpenAM, successfully, you will receive a valid SSO token.

The next time you access OpenIDM, with the SSO token stored in your browser, you will be logged in automatically, with all the roles configured for your user account. This works only if your session is still active in OpenAM. For more information, see the OpenAM Administration Guide section on *Session Management*.

Note

You can review the contents of the SSO token by pointing a browser to the FQDN of the OpenIDM instance, with the `info/login` endpoint. In this case, that URL would be <https://openidm.example.com/openidm/info/login>.

3.16.6. Additional Options for the OpenAM Sample

You can extend the OpenAM sample in one of three ways:

- `Disable INTERNAL_USER when enabling LiveSync`

To keep OpenDJ in sync with the OpenIDM managed/user database, you can enable scheduled reconciliation and LiveSync. To do so, edit the following files: `schedule-recon.json` and `schedule-livesync.json`, in the `samples/openam/conf` directory. Set `"enabled" : true` in each of these files.

This ensures that any change to the OpenDJ user store is automatically reflected in the OpenIDM managed/user data store. For a similar example, see Section 3.12, "Sample 6 - LiveSync With an AD Server".

You can take these steps while OpenIDM is running.

- Execute the reconciliation command described earlier in the description for this sample.
- Once scheduled reconciliation and LiveSync are configured, you can then disable the `INTERNAL_USER` under `authModules` in the relevant `authentication.json` file. Just navigate to that section of the file and set `"enabled" : false`. Once complete, the only users with access to OpenIDM are users with a valid SSO authentication token in their browsers.

3.17. Sample - Demonstrate Extended Audit Capabilities

This sample demonstrates how you can use the audit features associated with OpenIDM, for access, activity, reconciliation, and synchronization.

This sample uses a MySQL database as a target repository for audit information. This audit sample is closely related to the sample described in Chapter 2, "First OpenIDM Sample".

One difference: the audit sample includes a ScriptedSQL implementation of the Groovy Connector Toolkit.

If you want to use any of the audit features demonstrated in this sample, copy information from files in the `/path/to/samples/audit-sample` directory. This can help you set up auditing for any other sample.

3.17.1. Audit Sample Configuration Files

Review the configuration files used in this sample. They can help you understand the functionality of the data sets being audited.

The key configuration files, in the `/path/to/samples/audit-sample` directory, are as follows:

- `conf/provisioner.openicf-scriptedsql.json` shows the configuration of the Scripted SQL Connector (see Section 11.5.5, "Scripted SQL Connector" in the *Integrator's Guide*).
- `conf/provisioner.openicf-xml.json` shows the configuration of the XML File Connector (see Section 11.5.1, "XML File Connector" in the *Integrator's Guide*).
- `conf/audit.json` configures audit logging on the router to a remote system, as described in Section 18.3, "Audit Configuration" in the *Integrator's Guide*.
- `conf/sync.json` shows mappings between managed users and the data set attached through the XML File Connector.
- `data/sample_audit_db.mysql` includes a schema that supports tables in the external MySQL database.
- Groovy scripts in the `tools/` subdirectory supports communications between the Scripted SQL connector and the MySQL database.

3.17.2. Configuration with MySQL

You need to set up communications between OpenIDM and an external MySQL database server.

Make sure MySQL is running.

The sample expects the following configuration for MySQL:

- The database is available on the local system.

- The database listens on the standard MySQL port, 3306.
- You can connect to the MySQL database over the network.
- MySQL is configured with an administrative account with user `root` and password `password`.
- MySQL serves a database called `audit` with three tables: `access`, `activity`, and `recon`.
- For more information on the database schema, examine the following data definition language file: `openidm/samples/audit-sample/data/sample_audit_db.mysql`. Import the file into MySQL before running the sample.

```
$ mysql -u root -p < /path/to/openidm/samples/audit-sample/data/sample_audit_db.mysql
Enter password:
$
```

Now you can review the format of the audit database sample, created from the `sample_audit_db.mysql` file, at the MySQL prompt." access that prompt, run the following command:

```
$ mysql -u root -p
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
```

You can now review the current state of the access, activity, and reconciliation logs with the following commands:

```
select * from auditaccess;
select * from auditactivity;
select * from auditrecon;
select * from auditsync;
```

Unless you enable scheduled reconciliation, you will not see audit data until you run reconciliation manually.

Download the [MySQL Driver](#), (MySQL Connector/J, version 5.1 or later) from the MySQL website. Unpack the download and copy the `.jar` into the `openidm/bundle` directory.

```
$ cp mysql-connector-java-version-bin.jar \
/path/to/openidm/bundle/
```

3.17.3. Install the Sample

Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM", then start OpenIDM with the configuration for the audit sample.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/audit-sample
```

3.17.4. Running the Sample

Run reconciliation over the REST interface.

```
$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --header "Content-Type: application/json" \
  --request POST \
  "https://localhost:8443/openidm/recon?
_action=recon&mapping=systemXmlfileAccounts_managedUser&waitForCompletion=true"
```

Alternatively, open the `schedule-reconcile_systemXmlAccounts_managedUser.json` file in the `/path/to/openidm/samples/audit-sample/conf` directory and change the value of the `"enabled"` directive to turn on scheduled reconciliation:

```
"enabled" : true,
```

You can now review the results in the MySQL database, from the MySQL prompt, using the commands described earlier.

If you have retained the default `"useForQueries" : true` option in the `conf/audit.json` file, you can also **GET** the same data with a REST call. For examples on how you can query reconciliation, activity, and synchronization audit logs, see Chapter 12, *"Configuring Synchronization"* in the *Integrator's Guide*. For more information on the `useForQueries` property, see Section 18.3.2.1, "Logging to a Remote System" in the *Integrator's Guide*.

3.18. Sample - Connecting to Google With the Google Apps Connector

OpenICF 1.4.1.0 provides a new Google Apps Connector that enables you to interact with Google's web applications.

Note

The Google Apps Connector, and this corresponding sample, are provided only with the OpenIDM Enterprise release, available from ForgeRock's Backstage site.

This sample demonstrates the creation of users and groups on an external Google system, using OpenIDM's REST interface.

3.18.1. Before You Start

This sample requires that you have a Google Apps account. Obtaining a Google Apps account is described in the [Google documentation](#). The sample uses OAuth2 to authorize the connection to the

Google service. This authorization mechanism requires that you obtain a refresh token from Google. The following steps indicate how to obtain the refresh token.

1. Log in to your Google Apps Admin Console (at <https://www.google.com/a/domain-name>) and verify that the following APIs are enabled:
 - Admin SDK API
 - Enterprise License Manager API
2. Use the OAuth 2.0 Installed Application flow to obtain a `client_secrets.json` file. For more information about the OAuth 2.0 Installed Application flow, see the corresponding Google documentation.
3. Download the Google Admin Directory API client library (`google-api-services-admin-directory_v1-rev42-java-1.19.0.zip`) and extract it to a writeable directory.

```
$ unzip ~/Downloads/google-api-services-admin-directory_v1-rev42-java-1.19.0.zip -d /path/to
Archive:  ~/Downloads/google-api-services-admin-directory_v1-rev42-java-1.19.0.zip
  creating: /path/to/admin/
  extracting: /path/to/admin/google-api-services-admin-directory_v1-rev42-1.19.0.jar
  extracting: /path/to/admin/google-api-services-admin-directory_v1-rev42-1.19.0-javadoc
.jar
...
```

4. Download the Google Apps connector jar and copy it to the `admin` directory, created in the previous step.

```
$ cp ~/Downloads/googleapps-connector-1.4.1.0.jar /path/to/admin
```

5. Change to the `admin` directory and run the following command on the `client_secrets.json` file that you obtained earlier in this procedure.

```
$ cd admin/
$ java -jar googleapps-connector-1.4.0.0.jar /path/to/client_secrets.json
Please open the following address in your browser:
https://accounts.google.com/o/oauth2/auth?
access_type=offline
...
```

This command opens the default browser, and loads a screen on which you authorize consent to access the Google Apps account.

Tip

If you have recently created your Google Apps account, it might take some time (often two hours or more) to synchronize the data required for this access request to work.

6. When you have authorized consent, the browser returns a code. Copy and paste the code into the terminal from which you ran the original command.


```
Attempting to open that address in the default browser now...
Please enter code:XXXXXXXX
```

A response similar to the following is returned:

```
{
  "clientId" : "5x4x3x4x0x8x-cx1x3xsxcx8xixLxmx3x0xrxgx7x6x3x.apps.googleusercontent.com",
  "clientSecret" : "0xhx9xrx8xdxqx9xDxjxUx3x",
  "refreshToken" : "1x7xmxfx_yxuxNxUxFxjxVxVxkxXx3XxHxMxYxzx5xcxI"
}
```

- Use the information in this response to edit the `"configurationProperties"` in the Google Apps provisioner configuration file (`samples/google-connector/conf/provisioner.openicf-google.json`). For example:

```
"configurationProperties": {
  "domain": "example.com",
  "clientId": "5x4x3x4x0x8x-cx1x3xsxcx8xixLxmx3x0xrxgx7x6x3x.apps.googleusercontent.com",
  "clientSecret": "0xhx9xrx8xdxqx9xDxjxUx3x",
  "refreshToken": "1x7xmxfx_yxuxNxUxFxjxVxVxkxXx3XxHxMxYxzx5xcxI"
},
```

3.18.2. Running the Google Apps Sample

- To run the sample, start OpenIDM with the configuration for the Google Apps Connector.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/google-connector
Executing ./startup.sh...
Using OPENIDM_HOME: /path/to/openidm
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/samples/google-connector/conf/boot/boot.properties
OpenIDM version "3.1.0" (revision: 0) null
null
-> OpenIDM ready
```

- To verify the connector configuration, check the status of the Google Apps connector as follows:

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/system/google?_action=test"
{
  "name":"google",
  "enabled":true,
  "config":"config/provisioner.openicf/google",
  "connectorRef": {
    "connectorName":"org.forgerock.openicf.connectors.googleapps.GoogleAppsConnector",
    "bundleName":"org.forgerock.openicf.connectors.googleapps-connector",
    "bundleVersion":"[1.4.0.0,2.0.0.0]"
  },
  "ok":true
}
```

- To test that the connector is working, create, update, and read a user and group entry on the Google resource, over REST.

Note

When creating 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 \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "__NAME__": "samcarter@example.com",
  "__PASSWORD__": "password",
  "givenName": "Sam",
  "familyName": "Carter",
  "agreedToTerms": true,
  "changePasswordAtNextLogin": false
}' \
"https://localhost:8443/openidm/system/google/___ACCOUNT___?_action=create" \

{
  "phones":null,
  "givenName":"Sam",
  "familyName":"Carter",
  "thumbnailPhotoUrl":null,
  "addresses":null,
  "___NAME___":"samcarter@example.com",
  "changePasswordAtNextLogin":false,
  "suspensionReason":null,
  "aliases":null,
  "customerId":"Cwev45or",
}
```

```

"isDelegatedAdmin":false,
"lastLoginTime":["1970-01-01T00:00:00.000Z"],
"agreedToTerms":true,
"deletionTime":null,
"isAdmin":false,
"creationTime":["2014-09-23T18:54:19.000Z"],
"ims":null,
"organizations":null,
"ipWhitelisted":false,
"fullName":"Sam Carter",
"includeInGlobalAddressList":true,
"isMailboxSetup":false,
"externalIds":null,
"suspended":false,
"nonEditableAliases":null,
"orgUnitPath":"/",
"relations":null,
"emails":[
  {
    "address":"samcarter@example.com",
    "primary":true
  }
],
"_id":"101643010677712061243",
"_rev":"\SHoRIPAZisvsSqsLJr3m3XS9Uw4/RfZ0wrjqrpRs06PCGNfEp2H00Ds\"
}

```

4. Update the user's phone number by sending a PUT request with the updated data, and specifying the user ID in the request, for example:

```

$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "__NAME__": "samcarter@example.com",
  "__PASSWORD__": "password",
  "givenName": "Sam",
  "familyName": "Carter",
  "agreedToTerms": true,
  "changePasswordAtNextLogin": false
  "phones":
  [
    {
      "value": "1234567890",
      "type": "home"
    },
    {
      "value": "0987654321",
      "type": "work"
    }
  ]
}' \
"https://localhost:8443/openidm/system/google/___ACCOUNT___/101643010677712061243"

```

The response returns the complete user object, with the updated phone numbers.

5. Read the user entry from the Google resource:

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "If-None-Match" : "*" \
--request GET \
"https://localhost:8443/openidm/system/google/__ACCOUNT__/101643010677712061243"
{
  "phones":
  [
    {"value": "1234567890", "type": "home"},
    {"value": "0987654321", "type": "work"}
  ],
  "givenName": "Sam",
  "familyName": "Carter"
,
...
  "_id": "101643010677712061243",
  "_rev": "\SHoRIPAZisvsSqsLJr3m3XS9Uw4/RfZ0wrjqrpRs06PCGNfEp2H00Ds\""}
}
```

6. Create a group entry on the Google resource.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "__NAME__": "testGroup@example.com",
  "__DESCRIPTION__": "Group used for google-connector sample.",
  "name": "TestGroup"
}' \
"https://localhost:8443/openidm/system/google/__GROUP__?_action=create"
{
  "__DESCRIPTION__": "Group used for google-connector sample.",
  "directMembersCount": 0,
  "nonEditableAliases": null,
  "aliases": null,
  "adminCreated": true,
  "name": "TestGroup",
  "__NAME__": "testGroup@example.com",
  "_id": "03oy7u291s3qwli",
  "_rev": "\SHoRIPAZisvsSqsLJr3m3XS9Uw4/8JZsrHGxtaH4sF8vgzLiPmPqFs\""}
}
```

7. Add the user Sam Carter, that you created previously, to the test group. Include the `_id` that was generated in the user creation step in the request URL.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "groupKey": "03oy7u291s3qw1i",
  "role": "MEMBER",
  "__NAME__": "samcarter@example.com",
  "email": "samcarter@example.com",
  "type": "MEMBER"
}' \
"https://localhost:8443/openidm/system/google/Member/101643010677712061243"

{
  "role": "MEMBER",
  "type": "USER",
  "email": "samcarter@example.com",
  "__NAME__": "03oy7u291s3qw1i/samcarter@example.com",
  "groupKey": "109384837631312225274",
  "_id": "014ykbeg40r6x06/samcarter@example.com",
  "_rev": "\"SHoRIPAZisvsSqsLJr3m3XS9Uw4/GiZLiMKb3U62M58IabSzPKAVIOE\""
}
```

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

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/system/google/___GROUP___/03oy7u291s3qw1i"

{
  "__DESCRIPTION__": "Group used for google-connector sample.",
  "directMembersCount": 1,
  "nonEditableAliases": [
    "googleAppsTestGroup@example.com.test-google-a.com"
  ],
  "aliases": null,
  "adminCreated": true,
  "name": "TestGroup",
  "__NAME__": "testGroup@example.com",
  "_id": "03oy7u291s3qw1i",
  "_rev": "\"SHoRIPAZisvsSqsLJr3m3XS9Uw4/WcZC54etzuPtruE7uByf9NQ820w\""
}
```

9. Delete the group entry.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request DELETE \
"https://localhost:8443/openidm/system/google/__GROUP__/03oy7u291s3qw1i"

{
  "_id": "03oy7u291s3qw1i"
}
```

The delete request returns the group ID.

10. Delete the user Sam Carter, to return the Google system to its original state.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request DELETE \
"https://localhost:8443/openidm/system/google/__ACCOUNT__/101643010677712061243"

{
  "_id": "101643010677712061243"
}
```

3.19. Sample - Connecting to Salesforce With the Salesforce Connector

OpenIDM 3.1 provides a Salesforce Connector that enables provisioning, reconciliation, and synchronization with a Salesforce organization. The Salesforce Connector is not an OpenICF connector, but a separate OpenIDM module, based on the ForgeRock Common Resource API.

Note

The Salesforce Connector, and this corresponding sample, are provided only with the OpenIDM Enterprise release, available from ForgeRock's Backstage site.

This sample demonstrates the creation and update of users from OpenIDM to Salesforce, and from Salesforce to OpenIDM. You can use either the Admin UI, or the command line to run this sample. Both methods are outlined in the sections that follow.

3.19.1. Before you Start

This sample requires that you have a Salesforce account, and a Connected App with OAuth enabled. For more information about Connected Apps, see the [Connected Apps Overview](#) in the Salesforce documentation.

To set up a Connected App for OpenIDM, follow these steps:

1. Log in to salesforce.com with your Salesforce credentials.
2. Click *Setup* in the top right corner.
3. In the left hand menu, under *Build*, expand the *Create* item and click *Apps*.
4. On the right hand panel, scroll down to *Connected Apps* and click *New*.
5. In the *New Connected App* panel, enter the following Basic Information:
 - *Connected App Name*. Enter a name that you will recognize as the OpenIDM App, for example, **OpenIDM**.
 - *API Name*. This field defaults to the Connected App Name, but you can change it. Note that the Application API Name can only contain underscores and alphanumeric characters. The name must be unique, begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.
 - *Contact Email*. Enter the email address of the person responsible for this Connected App within your organization.
6. Select *Enable OAuth Settings* and enter the following information:
 - *Callback URL*. Enter the OpenIDM URL, to which the requested OAuth token will be sent, for example **<https://localhost:8443/admin/oauth.html>**.
 - *Selected OAuth Scopes*. Click the *Add* button to add the following *Available Auth Scopes* to the *Selected OAuth Scopes* column:
 - Access and manage your data
 - Access your basic information
 - Perform requests on your behalf at any time

New Connected App

[Help for this Page](#)

To publish an app, you need to be using a Developer Edition organization with a namespace prefix chosen.

Basic Information

|= Required Inform

Connected App Name

API Name

Contact Email

Contact Phone

Logo Image URL
[Upload logo image](#) or [Choose one of our sample logos](#)

Icon URL
[Choose one of our sample logos](#)

Info URL

Description

API (Enable OAuth Settings)

Enable OAuth Settings

Callback URL

Use digital signatures

Selected OAuth Scopes	Available OAuth Scopes	Selected OAuth Scopes
<ul style="list-style-type: none"> Access and manage your Chatter data (chatter_api) Access custom permissions (custom_permissions) Allow access to your unique identifier (openid) Full access (full) Provide access to custom applications (visualforce) Provide access to your data via the Web (web) 	<p>Add</p> <input type="button" value="▶"/> <input type="button" value="◀"/> <p>Remove</p>	<ul style="list-style-type: none"> 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)

You can leave the remaining fields blank.

7. Click *Save* to create the new Connected App.
8. The next window displays your new Connected App.

Under the *API (Enable OAuth Settings)* item, the Consumer Key and a link to the Consumer Secret are displayed.

Click the link to reveal the Consumer Secret.

▼ API (Enable OAuth Settings)

Consumer Key	3MVG98dostKihXN7Is8Q0g5q1	PdB5f5ATwmaMuWxl	Consumer Secret	485	425
Selected OAuth Scopes	Access your basic information (id, profile, email, address, phone) Access and manage your data (api) Perform requests on your behalf at any time (refresh_token, offline_access)			Callback URL	https://localhost:8443/admin/oauth.html

OpenIDM requires the Consumer Key and Secret to obtain an access token and a refresh token for access to salesforce.com.

Copy and paste both the key and the secret into a file for use when you set up the sample.

- To demonstrate the reconciliation of users from Salesforce to OpenIDM, your Salesforce organization should contain at least a few users. Add these users now if your Salesforce organization is empty.

3.19.2. Install the Sample

Prepare OpenIDM as described in Section 3.1.2, "Preparing OpenIDM", then start OpenIDM with the configuration for the Salesforce sample.

```
$ cd /path/to/openidm
$ ./startup.sh -p samples/salesforce-connector
```

3.19.3. Running the Sample by Using the Admin UI

The Admin UI is the recommended way to test this sample.

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

The first time you log into the Admin UI, you are prompted to change your password. If you do not want to change your password at this time, simply click X to close this window, and continue with the sample.

The Resources tab shows the Salesforce connector, which is currently disabled.

Resources

+ New Connector + New Managed Object + New Mapping

Connectors [help ?](#)

Connectors are external systems, databases, directory services, and so on to be managed by OpenIDM.


Disabled

salesforce
Salesforce Connector

New Connector

2. Enable the Salesforce connector by completing the authentication details as follows. You will need the Consumer Key and Consumer Secret that you obtained in the previous section.
 - Click on the Salesforce connector to open the Edit Connector dialog.
 - Select True from the Enabled list.
 - Select Salesforce Connector from the Connector Type list.
 - Under Basic Connector Details select the Sandbox URL (for the purposes of testing the sample) and enter the Connector Key and Secret that you obtained in the previous section.
 - You can leave the default LiveSync details for now. Click Update to update the connector configuration.


Edit Connector help

General Details 

Connector Name

Enabled

Connector Type

Base Connector Details 

1. Be sure your application is configured to use this Callback URL
https://localhost:8443/admin/oauth.html
2. You must grant this application all of the following OAuth Scopes:
 - Access your basic information
 - Access and manage your data
 - Perform requests on your behalf at any time

Login URL
 Production Sandbox Custom

Consumer Key

Consumer Secret

[Need help locating your consumer key and secret?](#)

Note for new and recently-changed apps

When you create a new Connected App, or update your Connected App in Salesforce, it takes some time (at least five minutes) for the data to be propagated through all the Salesforce servers. If you are setting up OpenIDM with a Connected App that you have just created, or if you have recently changed the Callback URL of your Connected App, you might receive an error when you attempt to validate the consumer key and consumer secret. You should therefore wait at least five minutes after creating or updating your Connected App before you attempt to set up Identity Connect with that App.

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

Note

In the current OpenIDM release, an issue is occasionally seen where the system appears to time out while retrieving the refresh token from Salesforce, at this stage. If this happens, refresh your browser and attempt the connector setup again.

On the Resources tab, your Salesforce Connector should now be Active.


Resources

+ New Connector + New Managed Object + New Mapping

Connectors

Connectors are external systems, databases, directory services, and so on to be managed by OpenIDM. [help](#)

Active



salesforce
Salesforce Connector

+

New Connector

- To test the reconciliation process, select the Mappings tab.





This tab shows two configured mappings, one from Salesforce to the OpenIDM repository (**managed/user**) and one from the OpenIDM repository to Salesforce.

Resources
Mappings

Mapping List

[help](#)

[New Mapping](#) +

sourceSalesforceUser_managedUser		
<p>Source</p>  <p>system/salesforce/User salesforce</p>	<p>→</p>	<p>Target</p>  <p>managed/user managed</p>
<p>Reconciliation Status: Not yet Reconciled</p> <p style="text-align: right;"> Delete Edit </p>		
managedUser_sourceSalesforceUser		
<p>Source</p>  <p>managed/user managed</p>	<p>→</p>	<p>Target</p>  <p>system/salesforce/User salesforce</p>
<p>Reconciliation Status: Not yet Reconciled</p> <p style="text-align: right;"> Delete Edit </p>		

- Click anywhere on the first mapping and click Reconcile Now.

sourceSalesforceUser_managedUser



Source
system/salesforce/User
salesforce





Target
managed/user
managed

▶ Status: Not yet Reconciled
[Reconcile Now](#)
[help](#)

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

- Retrieve the users in the repository by clicking the User View link at the top right of the Admin UI.

This link opens the User View UI. If you did not change your password in the first step, you are prompted to change your password again. You can bypass this by simply clicking X to close the password prompt window.

- Select the Users tab.

Dashboard
Users

Users list

Add user
Reload Grid
Clear Filters

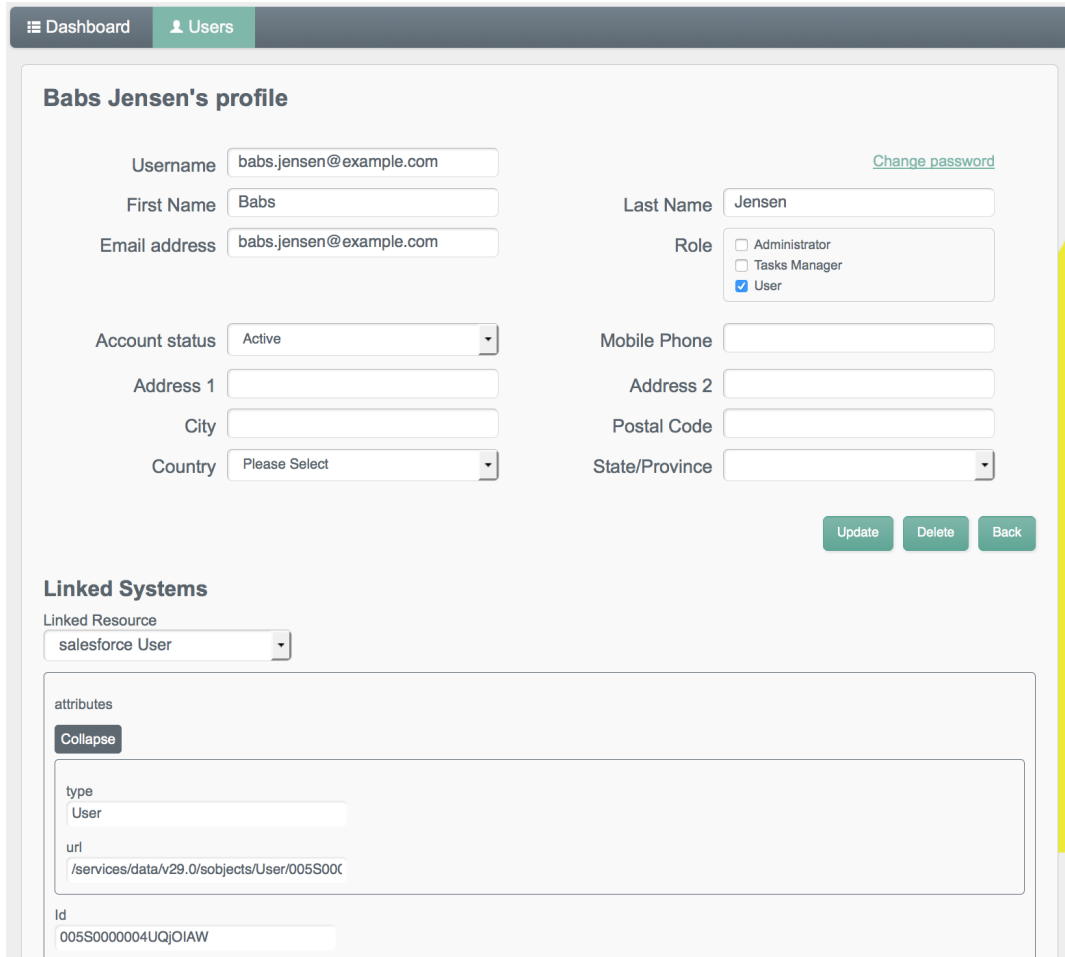
Username	First Name	Last Name	Email	Status
<input type="text" value="admin@identityconnect-dev.com"/> x	<input type="text" value="Admin"/> x	<input type="text" value="User"/> x	<input type="text" value="lana.frost@forgerock.com"/> x	All <input type="text" value=""/> x
admin@identityconnect-dev.com	Admin	User	lana.frost@forgerock.com	✔
babs.jensen@example.com	Babs	Jensen	babs.jensen@example.com	✔
ben.travis@example.com	Ben	Travis	ben.travis@example.com	✔
cindy.venter@example.com	Cindy	Venter	cindy.venter@example.com	✔
lerato.mmutle@example.com	Lerato	Mmutle	lerato.mmutle@example.com	✔
samantha.donnelly@example.com	Samantha	Donnelly	samantha.donnelly@example.com	✔
steven.carter@example.com	Steven	Carter	steven.carter@example.com	✔
zawadi.treffers@example.com	Zawadi	Treffers	zawadi.treffers@example.com	✔

Page 1 of 1
View 1 - 8 of 8

The users from the Salesforce organization have been reconciled to the OpenIDM repository. If the reconciliation was successful, the list of users displayed here should reflect what was in your Salesforce organization.

- To retrieve the details of a specific user, click that username on the Users tab.

The following image shows the details of user **bjensen**. Note the Linked Systems panel at the bottom of the image. This panel shows the corresponding user record in Salesforce.



Babs Jensen's profile

Username: [Change password](#)

First Name: Last Name:

Email address: Role: Administrator
 Tasks Manager
 User

Account status:

Mobile Phone:

Address 1: Address 2:

City: Postal Code:

Country: State/Province:

Linked Systems

Linked Resource:

attributes

type:

uri:

Id:

8. To test the second mapping (from OpenIDM to Salesforce), update any user in the OpenIDM repository. For example, update Babs Jensen's username.
9. 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 Section 12.3.2, "Synchronization Mappings File" in the *Integrator's Guide*.

To test that the implicit synchronization has been successful, look at Babs Jensen's record in the User View UI. At the bottom of the user profile, the Linked Systems panel indicates Babs Jensen's record in the Salesforce data store. Note the changed Username.

Alternatively, check the updated user record in Salesforce.

3.19.4. Running the Sample by Using the Command Line

Running the sample by using the command line is a little more complex. This section breaks the sample into two tasks - configuring the connector, and then testing the configuration by running reconciliation operations between the two systems.

Procedure 3.10. To Set Up the Salesforce Connector

Before you start, you will need the Consumer Key and Consumer Secret that you obtained in the previous section.

1. Obtain the refresh token from salesforce.com by pointing your browser to the following URL. Substitute your Consumer Key for `CLIENT_ID`. If OpenIDM is not running on the localhost, substitute the appropriate hostname and port number in the value of the `redirect_uri` parameter.

```
https://login.salesforce.com/services/oauth2/authorize?response_type=code&
client_id=CLIENT_ID&redirect_uri=https://localhost:8443/admin/oauth.html&scope=id%20api
%20refresh_token
```

2. You are redirected to Salesforce, and prompted to give this application access to your Salesforce account. When you have given consent, you should receive a response URL that looks similar to the following:

```
https://localhost:8443/admin/index.html#connectors/edit//
&code=aPrxJZTK7Rs03PU634VK8Jn9o_U3ZY1ERxM7IiklFzbnVpZ1TTJlU1TiETXKhNHqpr4SlyCkpg%3D%3D
```

The `&code` part of this URL is an authorization code, that you need for the following step.

Caution

Note that this authorization code expires after 10 minutes. If you do not complete the OAuth flow within that time, you will need to start this process again.

3. Copy the authorization code from the response URL and use it as the value of the `"code"` parameter in the following REST call. You will also need to supply your Consumer Key and Consumer Secret in this call.


```
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/system?_action=test"
[
  {
    "ok": true,
    "connectorRef": {
      "bundleVersion": "2.0.29.1a",
      "systemType": "provisioner.salesforce",
      "displayName": "Salesforce Connector",
      "bundleName": "org.forgerock.openidm.salesforce",
      "connectorName": "org.forgerock.openidm.salesforce.Salesforce"
    },
    "objectTypes": [
      "User",
      "PermissionSet",
      "PermissionSetAssignment",
      "Profile",
      "PermissionSetLicenseAssign",
      "Organization",
      "PermissionSetLicense",
      "Group",
      "GroupMember"
    ],
    "config": "config/provisioner.salesforce/salesforce",
    "enabled": true,
    "name": "salesforce"
  }
]
```

Procedure 3.11. To Run Reconciliation by Using the Command Line

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

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

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/recon?
_action=recon&mapping=sourceSalesforceUser_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 LDAP in the OpenIDM repository, assigning the new objects random unique IDs.

2. View the recon entry over REST for an indication of the actions that were taken on the OpenIDM repository.

```
$ curl \
--cacert self-signed.crt \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
"https://localhost:8443/openidm/recon/8a6281ef-6faf-43dd-af5c-3a842b38c468"
{
  "duration": 6447,
  "ended": "2014-11-28T15:01:38.399Z",
  "started": "2014-11-28T15:01:31.952Z",
  "parameters": {
    "null": false,
    "boolean": false,
    "number": false,
    "list": false,
    "object": {
      "targetQuery": {
        "_queryId": "query-all-ids",
        "resourceName": "managed/user"
      },
      "sourceQuery": {
        "_queryId": "query-all-ids",
        "resourceName": "system/salesforce/User"
      }
    }
  },
  "pointer": {
    "empty": true
  },
  "transformers": [],
  "set": false,
  "map": true,
  "string": false,
  "collection": false,
  "wrappedObject": {
    "targetQuery": {
      "resourceName": "managed/user",
      "_queryId": "query-all-ids"
    },
    "sourceQuery": {
      "_queryId": "query-all-ids",
      "resourceName": "system/salesforce/User"
    }
  }
},
  "_id": "8a6281ef-6faf-43dd-af5c-3a842b38c468",
  "mapping": "sourceSalesforceUser_managedUser",
  "state": "SUCCESS",
  "stage": "COMPLETED_SUCCESS",
  "stageDescription": "reconciliation completed.",
  "progress": {
```

```

"links": {
  "created": 8,
  "existing": {
    "total": "0",
    "processed": 0
  }
},
"target": {
  "created": 8,
  "existing": {
    "total": "0",
    "processed": 0
  }
},
"source": {
  "existing": {
    "total": "9",
    "processed": 9
  }
}
},
"situationSummary": {
  "FOUND_ALREADY_LINKED": 0,
  "UNASSIGNED": 0,
  "TARGET_IGNORED": 0,
  "SOURCE_IGNORED": 0,
  "MISSING": 0,
  "FOUND": 0,
  "AMBIGUOUS": 0,
  "UNQUALIFIED": 0,
  "CONFIRMED": 0,
  "SOURCE_MISSING": 0,
  "ABSENT": 9
},
"statusSummary": {
  "SUCCESS": 8,
  "FAILURE": 1
}
}

```

The output shows that eight entries were created on the target ([managed/user](#)).

3. You can display those users by querying the IDs in the [managed/user](#) repository.

```

$ curl \
  --cacert self-signed.crt \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --request GET \
  "https://localhost:8443/openidm/managed/user?_queryId=query-all-ids"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 8,
  "result": [
    {
      "_rev": "0",
      "_id": "f15322f2-5873-4e5f-a4e5-2d4bc03dd190"
    }
  ]
}

```

```
},
{
  "_rev": "0",
  "_id": "85879c60-afa1-4425-8c7a-5cccbaff587"
},
{
  "_rev": "0",
  "_id": "ed3fe655-29a6-4016-b6bc-4b2356911fd1"
},
{
  "_rev": "0",
  "_id": "34678464-c080-41b1-8da6-d5fde9d35aeb"
},
{
  "_rev": "0",
  "_id": "02d5da29-8349-4f35-affc-5f6c331307ef"
},
{
  "_rev": "0",
  "_id": "f91d6fce-bf27-4379-9411-fd626f8a9528"
},
{
  "_rev": "0",
  "_id": "6ace9220-59e7-4d97-8683-e03362a9150c"
},
{
  "_rev": "0",
  "_id": "56863eea-35d7-4aeb-a017-74ef28fd3116"
}
]
```

Chapter 4

Installing a Repository For Production

By default, OpenIDM uses OrientDB for its internal repository so that you do not have to install a database in order to evaluate OpenIDM. Before using OpenIDM in production, however, you must replace OrientDB with a supported JDBC repository.

OpenIDM 3.1 supports the use of MySQL, MS SQL, PostgreSQL, and Oracle Database as internal repositories. For details of the supported versions, see Chapter 2, "*Before You Install OpenIDM Software*" in the *Release Notes*.

For information about the general JDBC repository configuration, and how to map OpenIDM objects to JDBC database tables, see Chapter 5, "*Managing the OpenIDM Repository*" in the *Integrator's Guide*.

4.1. To Set Up OpenIDM With MySQL

After you have installed MySQL on the local host and *before starting OpenIDM for the first time*, set up OpenIDM to use the new repository, as described in the following sections.

This procedure assumes that a password has already been set for the MySQL root user.

1. 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/
```

2. Make sure that OpenIDM is stopped.

```
$ cd /path/to/openidm/  
$ ./shutdown.sh  
OpenIDM is not running, not stopping.
```

3. Remove `openidm/conf/repo.orientdb.json`.

```
$ cd /path/to/openidm/conf/  
$ rm repo.orientdb.json
```

4. Copy `openidm/db/mysql/conf/repo.jdbc.json` to the `openidm/conf` directory.

```
$ cd /path/to/openidm
$ cp db/mysql/conf/repo.jdbc.json conf/
```

5. Import the data definition language script for OpenIDM into MySQL.

```
$ cd /path/to/mysql
$ ./bin/mysql -u root -p < /path/to/openidm/db/mysql/scripts/openidm.sql
Enter password:
$
```

This step creates an `openidm` database for use as the internal repository, and a user `openidm` with password `openidm` who has all the required privileges to update the database.

```
$ ./bin/mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 5.5.19 MySQL Community Server
(GPL)
...
mysql> use openidm;
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_openidm |
+-----+
| auditaccess       |
| auditactivity     |
| auditrecon        |
| clusterobjectproperties |
| clusterobjects   |
| configobjectproperties |
| configobjects    |
| genericobjectproperties |
| genericobjects   |
| internaluser     |
| links             |
| managedobjectproperties |
| managedobjects   |
| objecttypes      |
| schedulerobjectproperties |
| schedulerobjects |
| security          |
| securitykeys     |
| uinotification   |
+-----+
19 rows in set (0.00 sec)
```

The table names are similar to those used with OrientDB.

6. Update `openidm/conf/repo.jdbc.json` as necessary, to reflect your MySQL deployment.

```

"connection" : {
  "dbType" : "MYSQL",
  "jndiName" : "",
  "driverClass" : "com.mysql.jdbc.Driver",
  "jdbcUrl" : "jdbc:mysql://localhost:3306/openidm",
  "username" : "openidm",
  "password" : "openidm",
  "defaultCatalog" : "openidm",
  "maxBatchSize" : 100,
  "maxTxRetry" : 5,
  "enableConnectionPool" : true,
  "connectionTimeoutInMs" : 30000
},
    
```

When you have set up MySQL for use as the OpenIDM internal repository, start OpenIDM to check that the setup has been successful. After startup, you should see that `repo.jdbc` is `active`, whereas `repo.orientdb` is `unsatisfied`.

```

$ cd /path/to/openidm
$ ./startup.sh
Using OPENIDM_HOME: /path/to/openidm
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/conf/boot/boot.properties
-> scr list

Id   State      Name
[ 19] [active    ] org.forgerock.openidm.config.starter
[ 23] [active    ] org.forgerock.openidm.taskscanner
[  8] [active    ] org.forgerock.openidm.external.rest
[ 12] [active    ] org.forgerock.openidm.provisioner.openicf.connectorinfoprovider
[ 15] [active    ] org.forgerock.openidm.ui.simple
[  1] [active    ] org.forgerock.openidm.router
[ 22] [active    ] org.forgerock.openidm.scheduler
[ 14] [active    ] org.forgerock.openidm.restlet
[  7] [unsatisfied] org.forgerock.openidm.external.email
[ 18] [unsatisfied] org.forgerock.openidm.repo.orientdb
[  6] [active    ] org.forgerock.openidm.sync
[  3] [active    ] org.forgerock.openidm.script
[  5] [active    ] org.forgerock.openidm.recon
[  2] [active    ] org.forgerock.openidm.scope
[ 10] [active    ] org.forgerock.openidm.http.contextregistrator
[ 20] [active    ] org.forgerock.openidm.config
[  0] [active    ] org.forgerock.openidm.audit
[ 21] [active    ] org.forgerock.openidm.schedule
[ 17] [active    ] org.forgerock.openidm.repo.jdbc
[ 16] [active    ] org.forgerock.openidm.workflow
[ 13] [active    ] org.forgerock.openidm.provisioner.openicf
[  4] [active    ] org.forgerock.openidm.managed
[  9] [active    ] org.forgerock.openidm.authentication
[ 11] [active    ] org.forgerock.openidm.provisioner
    
```

4.2. To Set Up OpenIDM With MS SQL

These instructions are specific to MS SQL Server 2008 running on a local Windows 2008 Server R2 system. Adapt the instructions for your environment.

When you install Microsoft SQL Server, note that OpenIDM has the following specific configuration requirements:

- During the Feature Selection installation step, make sure that at least SQL Server Replication, Full Text Search, and Management Tools - Basic are selected.

These instructions require SQL Management Studio so make sure that you include Management Tools in the installation.

- OpenIDM requires SQL Server authentication. During the MS SQL Server installation, make sure that you select SQL Server authentication (mixed mode) and not just Windows authentication.
- TCP/IP must be enabled and configured for the correct IP address and port. To configure TCP/IP, follow these steps:
 1. Click Start > All Programs > MS SQL Server 2008 R2 > Configuration Tools > SQL Server Configuration Manager.
 2. Expand the SQL Server Network Configuration item and select "Protocols for MSSQLSERVER".
 3. Double click TCP/IP and select Enabled > Yes.
 4. Select the IP Addresses tab and set the addresses and ports on which the server will listen.

For this sample procedure, scroll down to IPAll and set TCP Dynamic Ports to 1433 (the default port for MS SQL).

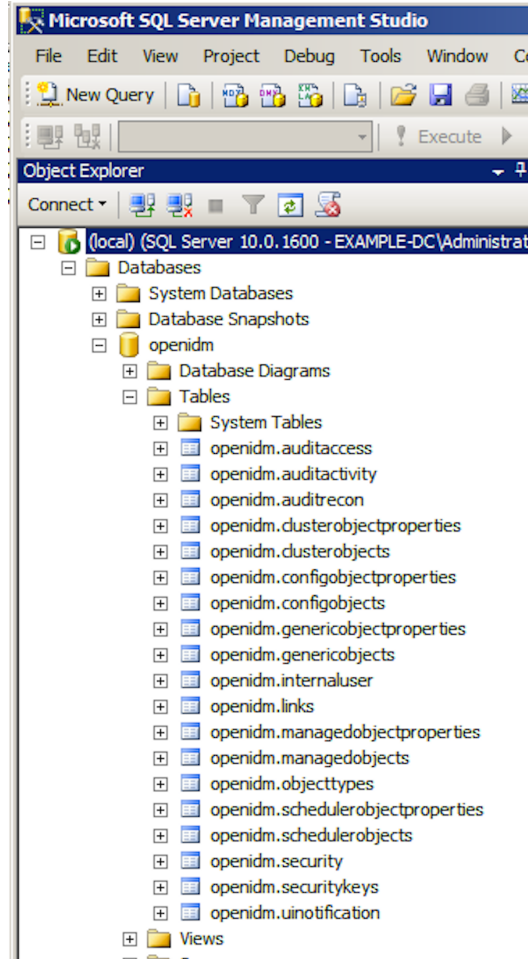
5. Click Apply, then OK.
6. Restart MS SQL Server for the configuration changes to take effect. To restart the server, select SQL Server Services in the left pane, double click SQL Server (MSSQLSERVER) and click Restart.
7. If you have a firewall enabled, ensure that the port you configured in the previous step is open for OpenIDM to access MS SQL.

After you have installed MS SQL on the local host, install OpenIDM, if you have not already done so, but *do not start* the OpenIDM instance. Import the data definition and set up OpenIDM to use the new repository, as described in the following steps.

1. Use SQL Management Studio to import the data definition language script for OpenIDM into MS SQL.
 - a. Click Start > All Programs > Microsoft SQL Server 2008 > SQL Server Management Studio.

- b. On the Connect to Server panel, under Server Type, select Database Engine. From the Authentication drop down list, select Windows Authentication. If needed, log in as the current user (for example, Administrator).
 - c. Select File > Open > File and navigate to the OpenIDM data definition language script (path `\to\openidm\db\mssql\scripts\openidm.sql`). Click Open to open the file.
 - d. Click Execute to run the script.
2. This step creates an `openidm` database for use as the internal repository, and a user `openidm` with password `Passw0rd` who has all the required privileges to update the database. You might need to refresh the view in SQL Server Management Studio to see the `openidm` database in the Object Explorer.

Expand Databases > `openidm` > Tables. You should see the OpenIDM tables in the `openidm` database, as shown in the following example:



The table names are similar to those used with OrientDB.

3. OpenIDM requires an MS SQL driver that must be created from two separate JAR files. Create the driver as follows.
 - a. Download the JDBC Driver 4.0 for SQL Server ([sqljdbc_4.0.2206.100_enu.exe](#)) from Microsoft's download site. The precise URL may vary, depending on your location.

Run the downloaded executable file; it should extract multiple files, include Java archive files, to a dedicated folder.

Extract the executable Java archive file ([sqljdbc4.jar](#)) from the dedicated folder, using 7-zip or an equivalent file management application.

Copy the Java archive file to `openidm\db\scripts\mssql`.

- b. Download the `bnd` Java archive file (`biz.aQute.bnd.jar`) that enables you to create OSGi bundles. The file can be downloaded from <http://dl.dropbox.com/u/2590603/bnd/biz.aQute.bnd.jar>. For more information about `bnd`, see <http://bnd.bndtools.org/>.

Copy the file to `openidm\db\mssql\scripts`.

- c. Your `openidm\db\mssql\scripts` directory should now contain the following files:

```
biz.aQute.bnd.jar  openidm.sql  sqljdbc4.bnd  sqljdbc4.jar
```

- d. Bundle the two JAR files together with the following command:

```
C:\> cd \path\to\openidm\db\mssql\scripts
./> java -jar biz.aQute.bnd.jar wrap -properties sqljdbc4.bnd sqljdbc4.jar
```

This step creates a single `.bar` file, named `sqljdbc4.bar`.

- e. Rename the `sqljdbc4.bar` file to `sqljdbc4-osgi.jar` and copy it to the `openidm\bundle` directory.

```
./> ren sqljdbc4.bar sqljdbc4-osgi.jar
./> copy sqljdbc4-osgi.jar \path\to\openidm\bundle
```

4. Remove the default OrientDB repository configuration file (`openidm\conf\repo.orientdb.json`) from the configuration directory.

```
C:\> cd \path\to\openidm\conf\
.\> del repo.orientdb.json
```

5. Copy the repository configuration file for MS SQL (`openidm\db\mssql\conf\repo.jdbc.json`) to the configuration directory.

```
C:\> cd \path\to\openidm
.\> cp db\mssql\conf\repo.jdbc.json conf\
```

6. Update `openidm\conf\repo.jdbc.json` as necessary, to reflect your MS SQL deployment.

```
{
  "connection" : {
    "dbType" : "SQLSERVER",
    "jndiName" : "",
    "driverClass" : "com.microsoft.sqlserver.jdbc.SQLServerDriver",
    "jdbcUrl" : "jdbc:sqlserver://localhost:1433;instanceName=default;
                databaseName=openidm;applicationName=OpenIDM",
    "username" : "openidm",
    "password" : "Passw0rd",
    "defaultCatalog" : "openidm",
    "maxBatchSize" : 100,
    "maxTxRetry" : 5,
    "enableConnectionPool" : true,
    "connectionTimeoutInMs" : 30000
  },
  ...
}
```

Specifically, check that the port matches what you have configured in MS SQL.

When you have completed the preceding steps, start OpenIDM to check that the setup has been successful. After startup, you should see that `repo.jdbc` is `active`, whereas `repo.orientdb` is `unsatisfied`.

```
C:> cd \path\to\openidm
./> startup.bat

"Using OPENIDM_HOME:   \path\to\openidm"
"Using OPENIDM_OPTS:  -Xmx1024m -Xms1024m -Dfile.encoding=UTF-8"
"Using LOGGING_CONFIG:
-Djava.util.logging.config.file=\path\to\openidm\conf\logging.properties"
Using boot properties at \path\to\openidm\conf\boot\boot
.properties
-> scr list
Id   State      Name
...
[ 18] [unsatisfied] org.forgerock.openidm.repo
.orientdb
...
[ 17] [active      ] org.forgerock.openidm.repo
.jdbc
...
```

4.3. To Set Up OpenIDM With Oracle Database

When implementing an Oracle database for OpenIDM, confer with an Oracle DBA when creating the database schema, tables, and users. This section assumes that you have configured an Oracle Database with *Local Naming Parameters* (`tnsnames.ora`) and a service user for use by OpenIDM.

Import the OpenIDM schema using the data definition language script (`/path/to/openidm/db/oracle/scripts/openidm.sql`), as the appropriate schema owner user.

If you have created OpenIDM tables correctly, you should be able to query the `internaluser` table. The query should return two records (`openidm-admin` and `anonymous`). The output here has been formatted for legibility.

```
SQL> select * from internaluser;

OBJECTID      openidm-admin
-----
REV           0
-----
PWD           openidm-admin
-----
ROLES         openidm-admin,openidm-authorized
-----

OBJECTID      anonymous
-----
REV           0
-----
PWD           anonymous
-----
ROLES         openidm-reg
-----
```

Before you start OpenIDM, create an Oracle DB driver from two separate jar files and set up the OpenIDM repository configuration for Oracle DB, as follows:

1. Download the Oracle JDBC driver for your Oracle DB version from Oracle Technology Network and place it in the `openidm/db/oracle/scripts` directory.

```
$ ls /path/to/openidm/db/oracle/scripts
ojdbc6_g.jar  openidm.sql
```

2. Create a bind file and edit it to match the version information for your JDBC driver.

You can use the sample bind file located in `openidm/db/mssql/scripts`. Copy the bind file to the same location as the JDBC driver.

```
$ cd /path/to/openidm/db
$ cp mssql/scripts/sqljdbc4.bnd oracle/scripts
$ ls oracle/scripts
ojdbc6_g.jar  openidm.sql  sqljdbc4.bnd
```

The JDBC driver version information for your driver is located in the `Specification-Version` property in the MANIFEST file of the driver.

```
$ cd /path/to/openidm/db/oracle/scripts
$ unzip -q -c ojdbc6_g.jar META-INF/MANIFEST.MF
...
Specification-Vendor: Sun Microsystems Inc.
Specification-Title: JDBC
Specification-Version: 4.0
...
```

Edit the bind file to match the JDBC driver version.

```
$ more sqljdbc4.bnd
...
version=4.0
Export-Package: *;version=${version}
Bundle-Name: Oracle JDBC Driver 4.0 for SQL Server
Bundle-SymbolicName: Oracle JDBC Driver 4.0 for SQL Server
Bundle-Version: ${version}
```

3. Download the `bnd` Java archive file (`biz.aQute.bnd.jar`) that enables you to create OSGi bundles. The file can be downloaded from <http://dl.dropbox.com/u/2590603/bnd/biz.aQute.bnd.jar>. For more information about `bnd`, see <http://bnd.bndtools.org/>.

Place the `bnd` Java archive file in the same directory as the JDBC driver, and the `bind` file.

```
$ ls /path/to/openidm/db/oracle/scripts
biz.aQute.bnd.jar  ojdbc6_g.jar  openidm.sql  sqljdbc4.bnd
```

4. Change to the directory in which the script files are located and run the following command to create the OSGi bundle.

```
$ cd /path/to/openidm/db/oracle/scripts
$ java -jar biz.aQute.bnd.jar wrap -properties sqljdbc4.bnd ojdbc6_g.jar
Dec 10, 2013 9:53:28 AM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
ojdbc6_g 984 0
```

A new `.bar` file has now been created.

```
$ ls
biz.aQute.bnd.jar  ojdbc6_g.bar  ojdbc6_g.jar  openidm.sql  sqljdbc4.bnd
```

5. Move the `.bar` file to the `openidm/bundle` directory and rename it with a `.jar` extension. The actual name of the file is unimportant.

```
$ mv ojdbc6_g.bar /path/to/openidm/bundle/ojdbc6_g-osgi.jar
```

6. Remove the default OrientDB configuration file (`openidm/conf/repo.orientdb.json`) from the configuration directory.

```
$ rm /path/to/openidm/conf/repo.orientdb.json
```

7. Copy the OracleDB configuration file (`openidm/db/oracle/conf/repo.jdbc.json`) to the configuration directory.

```
$ cd /path/to/openidm
$ cp db/oracle/conf/repo.jdbc.json conf/
```

8. Update `openidm/conf/repo.jdbc.json` as necessary, to reflect your OracleDB deployment. Specifically, edit the `jdbcUrl`, `username`, and `password` properties. For example:

```
"connection" : {
  "dbType" : "ORACLE",
  "jndiName" : "",
  "driverClass" : "oracle.jdbc.OracleDriver",
  "jdbcUrl" : "jdbc:oracle:thin:@//localhost:1521/openidm",
  "username" : "openidm",
  "password" : "password",
  "defaultCatalog" : "openidm",
  "maxBatchSize" : 100,
  "maxTxRetry" : 5,
  "connectionTimeoutInMs" : 30000
},
```

The following parameters relate to the Oracle database:

- The `"dbType"` is `"ORACLE"`.
- The `"driverClass"` is `"oracle.jdbc.OracleDriver"`.
- The `"jdbcUrl"` corresponds to the URL of the Oracle DB listener, including the service name, based on your configured Local Naming Parameters (tnsnames.ora). It should be whatever is appropriate for your environment. Replace `"openidm"` with the service name from your TNS description.
- The `"username"` and `"password"` corresponds to the credentials of the service user that connects from OpenIDM.
- The `"defaultCatalog"` is used to generate queries appropriate to your OpenIDM schema in Oracle. It should match the user who "owns" the tables. If your schema owner was `"openidm"`, then the `"defaultCatalog"` should also be `"openidm"`. This will cause OpenIDM to generate queries such as `"SELECT objectid FROM openidm.internaluser"`.

When you have set up OracleDB for use as the OpenIDM internal repository, start OpenIDM to check that the setup has been successful. On startup, a number of INFO messages are output, as the predefined queries are processed.

After startup, you should see that `repo.jdbc` is `active`, whereas `repo.orientdb` is `unsatisfied`.

```
$ cd /path/to/openidm
$ ./startup.sh
Using OPENIDM_HOME: /path/to/openidm
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/conf/boot/boot
.properties
....
-> scr list
   Id   State      Name
...
[  2] [unsatisfied] org.forgerock.openidm.repo
.orientdb
...
[  3] [active      ] org.forgerock.openidm.repo
.jdbc
...
```

4.4. To Set Up OpenIDM With PostgreSQL

Note

The use of PostgreSQL as a repository is supported for Java 1.7 only.

This procedure assumes that PostgreSQL (version 9.3 or later) is installed and running on the local host.

Before starting OpenIDM for the first time, set up OpenIDM to use a PostgreSQL repository, as described in the following procedure.

1. OpenIDM includes a script (`/path/to/openidm/db/postgresql/scripts/createuser.pgsql`) that sets up an `openidm` database and user, with a default password of `openidm`. The script also grants the appropriate permissions.

Edit this script if you want to change the password of the `openidm` user, for example:

```
$ more /path/to/openidm/db/postgresql/scripts/createuser.pgsql
create user openidm with password 'mypassword';
create database openidm encoding 'utf8' owner openidm;
grant all privileges on database openidm to openidm;
```

2. Execute the `createuser.pgsql` script as follows:

```
$ psql < /path/to/openidm/db/postgresql/scripts/createuser.pgsql
CREATE ROLE
CREATE DATABASE
GRANT
```

3. Execute the `openidm.pgsql` script as the new `openidm` user that you created in the first step.

```
$ psql -U openidm < /path/to/openidm/db/postgresql/scripts/openidm.pgsql

CREATE SCHEMA
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE INDEX
CREATE INDEX
...
START TRANSACTION
INSERT 0 1
INSERT 0 1
COMMIT
CREATE INDEX
CREATE INDEX
```

Your database has now been initialized.

4. Remove the OrientDB repository configuration file from the OpenIDM configuration directory (`conf/repo.orientdb.json`).

```
$ rm /path/to/openidm/conf/repo.orientdb.json
```


- Copy the PostgreSQL repository configuration file (`openidm/db/postgres/conf/repo.jdbc.json`) to the configuration directory.

```
$ cd /path/to/openidm
$ cp db/postgres/conf/repo.jdbc.json conf/
```

- If you changed the password in step 1 of this procedure, edit the `repo.jdbc.json` file to set the value for the `"password"` field to whatever password you set for the `openidm` user. For example:

```
$ more conf/repo.jdbc.json
{
  "connection" : {
    "dbType" : "POSTGRESQL",
    "jndiName" : "",
    "driverClass" : "org.postgresql.Driver",
    "jdbcUrl" : "jdbc:postgresql://localhost:5432/openidm",
    "username" : "openidm",
    "password" : "mypassword"
  },
  ...}
```

- PostgreSQL is now set up for use as the OpenIDM internal repository.

Start OpenIDM to check that the setup has been successful. After startup, you should see that `repo.jdbc` is `active`, whereas `repo.orientdb` is `unsatisfied`.

```
-> OpenIDM ready
scr list
Id   State      Name
...
[ 4 ] [unsatisfied] org.forgerock.openidm.repo
.orientdb
...
[ 3 ] [active      ] org.forgerock.openidm.repo
.jdbc
..
->
```

- If you are using the default project configuration, run the `default_schema_optimization.pgsql` script to create the required indexes. The file includes extensive comments on the indexes that are being created.

```
$ psql -U openidm < /path/to/openidm/db/postgresql/scripts/default_schema_optimization.pgsql
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
```

Chapter 5

Removing and Moving OpenIDM Software

This chapter shows you how to uninstall OpenIDM software and to move an existing install to a different location.

Procedure 5.1. To Remove OpenIDM Software

1. (Optional) Stop OpenIDM services if they are running, by entering `shutdown` at the `->` prompt either on the command line, or on the System Information tab of the Felix console.

```
-> shutdown
```

2. Remove the file system directory where you installed OpenIDM software.

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

3. (Optional) If you use a JDBC database for the internal repository, you can drop the `openidm` database.

Procedure 5.2. To Move OpenIDM Software

If you want to move OpenIDM to a different directory, you do not have to uninstall and reinstall. To move an existing OpenIDM instance, follow these steps:

1. Shut down OpenIDM, as described in Procedure 1.3, "To Stop the OpenIDM Services".
2. Remove the `felix-cache` directory.

```
$ cd path/to/openidm  
$ rm -rf felix-cache
```

3. Move the files.

```
$ mv path/to/openidm path/to/new-openidm
```

4. Start OpenIDM in the new location.

```
$ cd path/to/new-openidm  
$ ./startup.sh
```

Chapter 6

Migrating to OpenIDM 3.1.0

The migration process is largely dependent on your particular deployment and on the extent to which you have customized OpenIDM. We recommend that you engage ForgeRock services for help in migrating an existing deployment.

This process assumes that you are migrating a deployment from OpenIDM 3.0.0 to OpenIDM 3.1.0. For documentation on migrating a deployment from OpenIDM 2.1.0 to 3.0.0, see *Migrating to OpenIDM 3.0.0* in the *OpenIDM 3.0.0 Installation Guide*.

The steps outlined in this section indicate how to preserve customizations, where possible, and take advantage of the new functionality offered in this release. However, you must be aware of the changes made in OpenIDM 3.1.0 that might affect your existing deployment. Therefore, before starting this process, see Chapter 4, "*OpenIDM Compatibility*" in the *Release Notes*, and adjust your scripts and clients accordingly.

OpenIDM 3.1.0 introduces an Administrative UI (Admin UI), which you can use to define the overall system configuration. You may consider the Admin UI as an option, especially for less complex deployments. For information on the new Admin UI, see Section 4.1, "Configuring OpenIDM from the Admin UI" in the *Integrator's Guide*.

To perform a basic migration to OpenIDM 3.1.0, follow these steps. For the purposes of this procedure, the path to the existing 3.0 instance is defined as `/path/to/openidm-3.0`. The path to the new instance is defined as `/path/to/openidm-3.1`.

1. Download and extract the OpenIDM 3.1.0 server.
2. Stop your existing OpenIDM 3.0.0 server, if it is running.

```
$ cd /path/to/openidm-3.0
$ ./shutdown.sh
Stopping OpenIDM (81491)
```

3. Back up your existing deployment by archiving the entire `openidm` directory.
4. On the OpenIDM 3.1.0 server, edit the `conf/boot/boot.properties` file to match any customizations that you made on your OpenIDM 3.0.0 server. Specifically, check the following elements:
 - In OpenIDM 3.0.0, port numbers were specified in the `conf/boot/boot.properties` and `conf/config.properties` files. In OpenIDM 3.1, the HTTP, HTTPS, and mutual authentication ports are specified in the `conf/boot/boot.properties` file. If you changed the default ports in your OpenIDM 3.0 deployment, make sure that the corresponding ports are specified in this file.

- Check that the keystore and truststore passwords match the current passwords for the keystore and truststore of your OpenIDM 3.0 deployment

Depending on the level of customization you have made in your current deployment, it might be simpler to start with your OpenIDM 3.0 `boot.properties` file, and copy all new settings from that file to the version associated with OpenIDM 3.1. However, as a best practice, we recommend that you keep all configuration customizations (including new properties and changed settings) in a single location in your configuration files. You can then copy and paste these changes as appropriate.

5. Copy the contents of your original `security/` folder to the new instance.

```
$ cd /path/to/openidm-3.1
$ cp -r /path/to/openidm-3.0/security .
```

Make sure to modify the `boot.properties` file for your project to point to the relative location of your `keystore.jceks` and `truststore` security files.

For example, if your `security/` folder is located at `/path/to/openidm-3.1/myproject/security`, edit the `boot.properties` file as follows:

```
$ less /path/to/openidm/myproject/conf/boot/boot.properties
...
openidm.keystore.type=JCEKS
openidm.truststore.type=JKS
openidm.keystore.provider=
openidm.keystore.location=myproject/security/keystore.jceks
openidm.truststore.location=myproject/security/truststore
```

OpenIDM automatically prepends the locations of the `keystore.jceks` and `truststore` files with the installation directory.

6. Migrate any custom scripts or default scripts that you have modified to the directory with OpenIDM 3.1. In general, custom and customized scripts should be located in the `openidm-3.0/script` directory on the OpenIDM 3.0 deployment.
 - For custom scripts, review [Chapter 4, "OpenIDM Compatibility"](#) in the *Release Notes* to ensure that the scripts will work as intended with the new version, then copy these scripts to the new instance. For example:

```
$ cd /path/to/openidm-3.1
$ cp /path/to/openidm-3.0/script/my-custom-script.js script/
```

- If you modified an existing OpenIDM 3.0 script, compare that modified script against the corresponding script in OpenIDM 3.1. If nothing has changed in the default script, check that your customizations will work as intended (by reviewing [Chapter 4, "OpenIDM Compatibility"](#) in the *Release Notes*) then copy the customized scripts to the new `openidm/script` directory. For example:

```
$ cd /path/to/openidm-3.1
$ cp /path/to/openidm-3.0/script/policy.js script/
```

- If a default script has changed since the 3.0 release, such as `linkedView.js`, copy the modified script to the `openidm-3.1/script` directory.

```
$ cd /path/to/openidm-3.0
$ cp bin/default/script/linkedView.js script/
```

Check that your customizations will work as expected, then port your customizations to the new script in the `openidm-3.1/script` directory.

7. Several changes have been made to the default configuration in OpenIDM 3.1.0. Currently, there is no automated way to migrate a customized configuration to the new version. The following strategy is recommended:
 - Start with the default 3.0.0 configuration.
 - For each configuration file that you have customized, use a file comparison tool such as the UNIX **diff** utility to assess the differences between your customized file and the 3.1 file.
 - Based on the results of the **diff**, use either your existing file as a base and port the 3.1.0 changes to that file, or vice versa. Ultimately, you want to preserve your customizations but ensure that you are up to date with the latest default configuration. All files should end up in the `openidm-3.1/conf` directory.
 - Pay particular attention to the `conf/repo.jdbc.json` file in your existing deployment. If you have customized this file, make sure that these customizations are ported to the corresponding file in the 3.1.0 deployment. For example, if you have defined any new queries, add these queries to the OpenIDM 3.1 instance of the `repo.jdbc.json` file.
8. Modify any customized provisioner configurations in your existing project to point to the new connectors that are provided with OpenIDM 3.1.0. Specifically, check that the `"connectorRef"` properties reflect the new connectors, for example:

```
{
  "bundleName": "org.forgerock.openicf.connectors.ldap-connector",
  "bundleVersion": "[1.4.0.0,2.0.0.0)",
  "displayName": "LDAP Connector",
  "connectorName": "org.identityconnectors.ldap.LdapConnector"
},
```

Alternatively, copy the connector jars from your existing installation into the `openidm/connectors/` folder of the new installation.

9. Complete the OpenIDM 3.1.0 installation, as described in Chapter 1, "Installing OpenIDM Services".
10. Migrate your internal user data, managed objects, and reconciliation and audit data, if required.

When you migrate this data, note the following points:

- The way in which queries on system objects are constructed has changed. This includes correlation queries on system objects. For more information, see Section 7.3.4, "Constructing Queries" in the *Integrator's Guide*.
- The database schema has changed slightly for OpenIDM 3.1.0:
 - The `auditsync` table has been added.
 - The `reconnection` column has been added to the `auditrecon` table.
 - The `userid` column has been added to the `auditaccess` table.

Your data migration strategy might vary, depending on your repository. You can either migrate your existing 3.0.0 database, or start with a new 3.1.0 database and import your existing data.

To migrate an existing database:

- Using the appropriate schema script from the new OpenIDM 3.1.0 instance (`/path/to/openidm-3.1/db/repo/scripts/openidm.sql`), take the changes described above and apply them to your existing database.

Use the `--force` option in MySQL (or an equivalent option for your repository type) to create the new tables, then edit the `auditsync` table manually, to add the columns described previously.

To start with a new database:

- Set up a clean repository, using the appropriate schema script from the new OpenIDM 3.1.0 instance (`/path/to/openidm-3.1/db/repo/scripts/openidm.sql`).
- Use a schema comparison tool and adjust the tables in your existing repository to match the schema in the new repository.
- Export your existing data to the new repository.

11. If you are using the UI, clear your browser cache after the migration. The browser cache contains files from the previous OpenIDM release, that might not be refreshed when you log into the new UI.

12. Start up OpenIDM 3.1.0.

```
$ cd /path/to/openidm-3.1
$ ./startup.sh
```

13. Test that your existing clients and scripts are working as intended.

OpenIDM Glossary

JSON	JavaScript Object Notation, a lightweight data interchange format based on a subset of JavaScript syntax. For more information, see the JSON site .
JWT	JSON Web Token. As noted in the <i>JSON Web Token draft IETF Memo</i> , "JSON Web Token (JWT) is a compact URL-safe means of representing claims to be transferred between two parties." For OpenIDM, the JWT is associated with the <code>JWT_SESSION</code> authentication module.
managed object	An object that represents the identity-related data managed by OpenIDM. Managed objects are configurable, JSON-based data structures that OpenIDM 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 a good introduction, see the OSGi site . OpenIDM services are designed to run in any OSGi container, but OpenIDM currently runs in Apache Felix .
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.
source object	In the context of reconciliation, a source object is a data object on the source system, that OpenIDM scans before attempting to find a corresponding object on the target system. Depending on the defined mapping, OpenIDM 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 OpenIDM for the period during which OpenIDM 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 OpenIDM scans after locating its corresponding object on the source system. Depending on the defined mapping, OpenIDM then adjusts the target object to match the corresponding source object.

Index

A

Application container
Requirements, 1

D

Downloading, 2

G

Getting started, 5, 11

I

Installing, 2
Samples, 29

J

Java
Requirements, 1

R

Repository database
Evaluation version, 11
Production ready, 151
Requirements, 2
Table names, 152

S

Samples
Google Apps Connector, 129
Microsoft Azure AD Connector, 86
Salesforce Connector, 136
Sample 1 - XML file, 11
Sample 2 - LDAP one way, 30
Sample 2b - LDAP two way, 36
Sample 2c - Synchronizing LDAP Group
Membership, 44
Sample 2d - Synchronizing LDAP Groups, 53
Sample 4 - CSV file, 101
Sample 5 - Synchronization of two resources,
102
Sample 5b - Failure Compensation With
Multiple Resources, 105

Sample 6 - LiveSync between two LDAP
servers, 107

Sample 7 - Scripting a SCIM-like Schema, 115

Sample 8 - Logging in Scripts, 117

Sample 9 - asynchronous reconciliation, 118

Sample Audit - Extended Audit Capabilities,
126

Sample OpenAM - Authentication Management
for OpenIDM, 121

ScriptedCREST Connector, 78

ScriptedREST Connector, 70

Starting OpenIDM, 3

Stopping OpenIDM, 5

U

Uninstalling, 164