Integrator's Guide

OpenIDM 2.1

Anders Askåsen
Paul Bryan
Mark Craig
Andi Egloff
Lana Frost
Laszlo Hordos
Matthias Tristl

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 configurating and integrating OpenIDM into identity management solutions. 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 vii
- 1. Who Should Use this Guide vii
- 2. Formatting Conventions vii
- 3. Accessing Documentation Online viii
- 4. Using the ForgeRock.org Site viii

1. Architectural Overview 1
- 1.1. OpenIDM Modular Framework 1
- 1.2. Infrastructure Modules 2
- 1.3. Core Services 3
- 1.4. Access Layer 4

2. Starting and Stopping OpenIDM 5
- 2.1. To Start and Stop OpenIDM 5
- 2.2. Specifying the OpenIDM Startup Configuration 6
- 2.3. Obtaining Information About an OpenIDM Instance 8
- 2.4. Verifying the Health of an OpenIDM System 10

3. OpenIDM Command-Line Interface 13
- 3.1. configexport 13
- 3.2. configimport 14
- 3.3. configureconnector 15
- 3.4. encrypt 16
- 3.5. keytool 18
- 3.6. validate 19

4. OpenIDM User Interface 21
- 4.1. Overview of the Default User Interface 21
- 4.2. Configuring the Default User Interface 22
- 4.3. Managing User Accounts With the User Interface 24
- 4.4. Managing Workflows From the User Interface 26

5. Configuring OpenIDM 27
- 5.1. OpenIDM Configuration Objects 27
- 5.2. Changing the Default Configuration 28
- 5.3. Configuring an OpenIDM System for Production 28
- 5.4. Configuring OpenIDM Over REST 29
- 5.5. Using Property Value Substitution in the Configuration 34
- 5.6. Adding Custom Endpoints 37

6. Accessing Data Objects 41
- 6.1. Accessing Data Objects by Using Scripts 41
- 6.2. Accessing Data Objects by Using the REST API 42
- 6.3. Defining and Calling Queries 42

7. Using Policies to Validate Data 44
- 7.1. Configuring the Default Policy 44
- 7.2. Extending the Policy Service 48
- 7.3. Disabling Policy Enforcement 49
- 7.4. Managing Policies Over REST 49

8. Configuring Server Logs 54

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

9. Connecting to External Resources ... 55
9.1. About OpenIDM & OpenICF ... 55
9.2. Accessing Remote Connectors ... 56
9.3. Configuring Connectors ... 58
9.4. Connector Configuration Examples ... 65
9.5. Creating Default Connector Configurations ... 90
10. Configuring Synchronization ... 97
10.1. Types of Synchronization ... 97
10.2. Managing Reconciliation Over REST ... 98
10.3. Triggering LiveSync Over REST ... 102
10.4. Flexible Data Model ... 103
10.5. Basic Data Flow Configuration ... 104
10.6. Synchronization Situations and Actions ... 112
10.7. Asynchronous Reconciliation ... 118
10.8. Configuring Case Sensitivity for Data Stores ... 119
10.9. Reconciliation Optimization ... 120
10.10. Correlation Queries ... 121
10.11. Advanced Data Flow Configuration ... 123
10.12. Scheduling Synchronization ... 126
11. Scheduling Tasks and Events ... 129
11.1. Scheduler Configuration ... 129
11.2. Configuring Persistent Schedules ... 133
11.3. Schedule Examples ... 134
11.4. Checking For Quartz Updates ... 135
11.5. Service Implementer Notes ... 135
11.6. Scanning Data to Trigger Tasks ... 135
12. Managing Passwords ... 141
12.1. Enforcing Password Policy ... 141
12.2. Password Synchronization ... 142
13. Managing Authentication, Authorization and RBAC ... 148
13.1. OpenIDM Users ... 148
13.2. Authentication ... 149
13.3. Roles ... 150
13.4. Authorization ... 151
14. Securing & Hardening OpenIDM ... 154
14.1. Use SSL and HTTPS ... 154
14.2. Restrict REST Access to the HTTPS Port ... 154
14.3. Encrypt Data Internally & Externally ... 157
14.4. Use Message Level Security ... 157
14.5. Replace Default Security Settings ... 159
14.6. Secure Jetty ... 160
14.7. Protect Sensitive REST Interface URLs ... 161
14.8. Protect Sensitive Files & Directories ... 162
14.9. Obfuscate Bootstrap Information ... 162
14.10. Remove or Protect Development & Debug Tools ... 162
14.11. Protect the OpenIDM Repository ... 162
14.12. Adjust Log Levels ... 163

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

14.13. Set Up Restart At System Boot 163
15. Integrating Business Processes and Workflows 164
15.1. BPMN 2.0 and the Activiti Tools 164
15.2. Setting Up Activiti Integration With OpenIDM 165
15.3. Managing Workflows Over the REST Interface 175
15.4. Example Activiti Workflows With OpenIDM 180
16. Using Audit Logs 188
16.1. Audit Log Types 188
16.2. Audit Log File Formats 189
16.3. Audit Configuration 192
16.4. Generating Reports 194
17. Sending Email 196
17.1. Sending Mail Over REST 197
17.2. Sending Mail From a Script 198
18. OpenIDM Project Best Practices 199
18.1. Implementation Phases 199
19. Troubleshooting 201
19.1. OpenIDM Stopped in Background 201
19.2. Internal Server Error During Reconciliation or Synchronization 201
19.3. The scr list Command Shows Sync Service As Unsatisfied 202
19.4. JSON Parsing Error 202
19.5. System Not Available 203
19.6. Bad Connector Host Reference in Provisioner Configuration 203
19.7. Missing Name Attribute 204
A. File Layout 205
B. Ports Used 212
C. Data Models and Objects Reference 213
C.1. Managed Objects 214
C.2. Configuration Objects 225
C.3. System Objects 227
C.4. Audit Objects 227
C.5. Links 228
D. Synchronization Reference 229
D.1. Object-Mapping Objects 229
D.2. Links 234
D.3. Queries 235
D.4. Reconciliation 235
D.5. REST API 236
E. REST API Reference 238
E.1. URI Scheme 238
E.2. Object Identifiers 238
E.3. Content Negotiation 238
E.4. Conditional Operations 239
E.5. Supported Methods 239
F. Scripting Reference 244
F.1. Scripting Configuration 244
F.2. Examples 245

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

F.3. Function Reference 245
F.4. Places to Trigger Scripts 255
F.5. Variables Available in Scripts 256
F.6. Debugging OpenIDM Scripts 257
G. Router Service Reference 259
G.1. Configuration 259
G.2. Example 263
H. Embedded Jetty Configuration 264
H.1. Using OpenIDM Configuration Properties in the Jetty Configuration 264
H.2. Jetty Default Settings 265
H.3. Registering Additional Servlet Filters 266
OpenIDM Glossary 268
Index 270

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Preface

This guide shows you how to integrate OpenIDM as part of a complete identity management solution.

1. Who Should Use this Guide

This guide is written for systems integrators building identity management solutions based on OpenIDM services. This guide describes OpenIDM, and shows you how to set up OpenIDM as part of your identity management solution.

You do not need to be an OpenIDM wizard to learn something from this guide, though a background in identity management and building identity management solutions can help.

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.");
    }
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

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

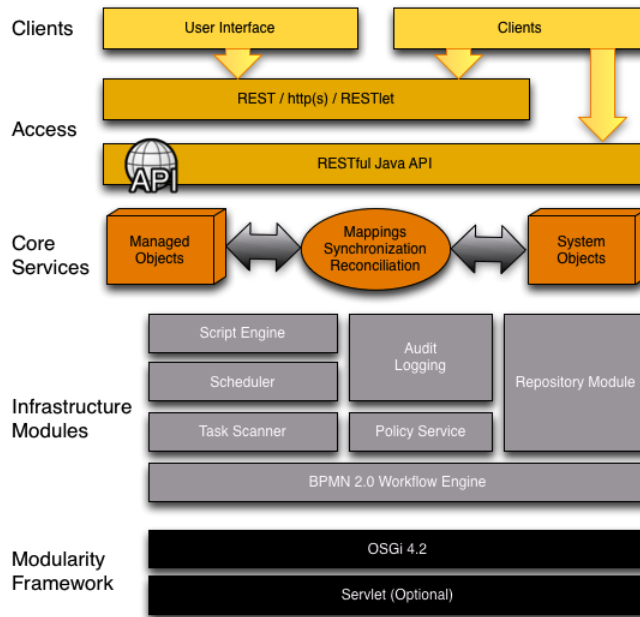
The ForgeRock.org site has links to source code for ForgeRock open source software, as well as links to the ForgeRock forums and technical blogs.

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.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 1
Architectural Overview

The following figure provides an overview of the OpenIDM architecture, which is covered in more detail in subsequent sections of this chapter.



1.1. OpenIDM Modular Framework

The OpenIDM framework is based on OSGi.

OSGi

OSGi is 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. While OpenIDM services are designed to run in any OSGi container, OpenIDM currently runs in Apache Felix.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Servlet

The optional Servlet layer provides RESTful HTTP access to the managed objects and services. While the Servlet layer can be provided by many different engines, OpenIDM embeds Jetty by default.

1.2. Infrastructure Modules

OpenIDM infrastructure modules provide the underlying features needed for core services.

BPMN 2.0 Workflow Engine

OpenIDM provides an embedded workflow and business process engine based on Activiti and the Business Process Model and Notation (BPMN) 2.0 standard.

For more information, see *Integrating Business Processes and Workflows*.

Task Scanner

OpenIDM provides a task scanning mechanism that enables you to perform a batch scan for a specified date in OpenIDM data, on a scheduled interval, and then to execute a task when this date is reached.

For more information, see *Scanning Data to Trigger Tasks*.

Scheduler

The scheduler provides a **cron**-like scheduling component implemented using the Quartz library. Use the scheduler, for example, to enable regular synchronizations and reconciliations.

See the *Scheduling Synchronization* chapter for details.

Script Engine

The script engine is a pluggable module that provides the triggers and plugin points for OpenIDM. OpenIDM currently implements a JavaScript engine.

Policy Service

OpenIDM provides an extensible policy service that enables you to apply specific validation requirements to various components and properties.

For more information, see *Using Policies to Validate Data*.

Audit Logging

Auditing logs all relevant system activity to the configured log stores. This includes the data from reconciliation as a basis for reporting, as well as detailed activity logs to capture operations on the internal (managed) and external (system) objects.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

See the *Using Audit Logs* chapter for details.

Repository

The repository provides a common abstraction for a pluggable persistence layer. OpenIDM 2.1 supports use of MySQL to back the repository. Yet, plugin repositories can include NoSQL and relational databases, LDAP, and even flat files. The repository API operates using a JSON-based object model with RESTful principles consistent with the other OpenIDM services. The default, embedded implementation for the repository is the NoSQL database OrientDB, making it easy to evaluate OpenIDM out of the box before using MySQL in your production environment.

1.3. Core Services

The core services are the heart of the OpenIDM resource oriented unified object model and architecture.

Object Model

Artifacts handled by OpenIDM are Java object representations of the JavaScript object model as defined by JSON. The object model supports interoperability and potential integration with many applications, services and programming languages. As OpenIDM is a Java-based product, these representations are instances of classes: `Map`, `List`, `String`, `Number`, `Boolean`, and `null`.

OpenIDM can serialize and deserialize these structures to and from JSON as required. OpenIDM also exposes a set of triggers and functions that system administrators can define in JavaScript which can natively read and modify these JSON-based object model structures. OpenIDM is designed to support other scripting and programming languages.

Managed Objects

A *managed object* is 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.

You can access managed objects over the REST interface with a query similar to the following:

```
$ curl
  --header "X-OpenIDM-Username: openidm-admin"
  --header "X-OpenIDM-Password: openidm-admin"
  --request GET
  "http://localhost:8080/openidm/managed/..."
```

System Objects

System objects are pluggable representations of objects on external systems. For example, a user entry that is stored in an external LDAP directory is represented as a system object in OpenIDM.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

System objects follow the same RESTful resource-based design principles as managed objects. They can be accessed over the REST interface with a query similar to the following:

```
$ curl
  --header "X-OpenIDM-Username: openidm-admin"
  --header "X-OpenIDM-Password: openidm-admin"
  --request GET
  "http://localhost:8080/openidm/system/..."
```

There is a default implementation for the OpenICF framework, that allows any connector object to be represented as a system object.

Mappings

Mappings define policies between source and target objects and their attributes during synchronization and reconciliation. Mappings can also define triggers for validation, customization, filtering, and transformation of source and target objects.

See the *Configuring Synchronization* chapter for details.

Synchronization & Reconciliation

Reconciliation provides for on-demand and scheduled resource comparisons between the OpenIDM managed object repository and source or target systems. Comparisons can result in different actions depending on the mappings defined between the systems.

Synchronization provides for creating, updating, and deleting resources from a source to a target system either on demand or according to a schedule.

See the *Configuring Synchronization* chapter for details.

1.4. Access Layer

The access layer provides the user interfaces and public APIs for accessing and managing the OpenIDM repository and its functions.

RESTful Interfaces

OpenIDM provides REST APIs for CRUD operations and invoking synchronization and reconciliation for both HTTP and Java.

See the *REST API Reference* appendix for details.

User Interfaces

User interfaces provide password management, registration, self-service, and workflow services.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 2
Starting and Stopping OpenIDM

This chapter covers the scripts provided for starting and stopping OpenIDM, and describes how to verify the *health* of a system, that is, that all requirements are met for a successful system startup.

2.1. To Start and Stop OpenIDM

By default you start and stop OpenIDM in interactive mode.

To start OpenIDM interactively, open a terminal or command window, change to the `openidm` directory, and run the startup script:

- **startup.sh** (UNIX)
- **startup.bat** (Windows)

The startup script starts OpenIDM, and opens an OSGi console with a `->` prompt where you can issue console commands.

To stop OpenIDM interactively in the OSGi console, enter the **shutdown** command.

```
-> shutdown
```

You can also start OpenIDM as a background process on UNIX, Linux, and Mac OS X. Follow these steps *before starting OpenIDM for the first time*.

1. If you have already started OpenIDM, then shut down OpenIDM and remove the Felix cache files under `openidm/felix-cache/`.

   ```
   -> shutdown
   ...
   $ rm -rf felix-cache/*
   ```

2. Disable `ConsoleHandler` logging before starting OpenIDM by editing `openidm/conf/logging.properties` to set `java.util.logging.ConsoleHandler.level = OFF`, and to comment out other references to `ConsoleHandler`, as shown in the following excerpt.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
# ConsoleHandler: A simple handler for writing formatted records to System.err
#handlers=java.util.logging.FileHandler, java.util.logging.ConsoleHandler
handlers=java.util.logging.FileHandler
...
# --- ConsoleHandler ---
# Default: java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.level = OFF
#java.util.logging.ConsoleHandler.formatter = ...
#java.util.logging.ConsoleHandler.filter=...
```

3. Remove the text-based OSGi console bundle, `bundle/org.apache.felix.shell.tui-version.jar`.
4. Start OpenIDM in the background.

   ```
   $ ./startup.sh &
   ```

   Alternatively, use the **nohup** command to keep OpenIDM running after you log out.

   ```
   $ nohup ./startup.sh &
   [2] 394
   $ appending output to nohup.out
   $
   ```

To stop OpenIDM running as a background process, use the **shutdown.sh** script.

```
$ ./shutdown.sh
./shutdown.sh
Stopping OpenIDM (454)
```

2.2. Specifying the OpenIDM Startup Configuration

By default, OpenIDM starts up with the configuration and script files that are located in the `openidm/conf` and `openidm/script` directories, and with the binaries that are in the default install location. You can launch OpenIDM with a different configuration and set of script files, and even with a different set of binaries, in order to test a new configuration, managed multiple different OpenIDM projects, or to run one of the included samples.

The `startup.sh` script enables you to specify the following elements of a running OpenIDM instance.

- project location (`-p`)

  The project location specifies the configuration and default scripts with which OpenIDM will run.

  If you specify the project location, OpenIDM does not try to locate configuration objects in the default location. All configuration objects and any artifacts that are not in the bundled defaults (such as custom scripts) *must* be provided in the project location. This includes everything that is in the default `openidm/conf` and `openidm/script` directories.

  The following command starts OpenIDM with the configuration of sample 1:

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


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

If an absolute path is not provided, the path is relative to the system property, `user.dir`. If no project location is specified, OpenIDM is launched with the default configuration in `/path/to/openidm/conf`.

- working location (`-w`)

  The working location specifies the directory to which OpenIDM writes its cache. Specifying a working location separates the project from the cached data that the system needs to store. The working location includes everything that is in the default `openidm/db` and `openidm/audit`, `openidm/felix-cache`, and `openidm/logs` directories.

  The following command specifies that OpenIDM writes its cached data to `/Users/admin/openidm/storage`:

  ```
  $ ./startup.sh -w /Users/admin/openidm/storage
  ```

  If an absolute path is not provided, the path is relative to the system property, `user.dir`. If no working location is specified, OpenIDM writes its cached data to `openidm/db` and `openidm/logs`.

- startup configuration file (`-c`)

  A customizable startup configuration file (named `launcher.json`) enables you to specify how the OSGi Framework is started.

  If no configuration file is specified, the default configuration (defined in `/path/to/openidm/bin/launcher.json`) is used. The following command starts OpenIDM with an alternative startup configuration file:

  ```
  $ ./startup.sh -c /Users/admin/openidm/bin/launcher.json
  ```

  You can modify the default startup configuration file to specify a different startup configuration.

  The customizable properties of the default startup configuration file are as follows:

  - `"location" : "bundle"` - resolves to the install location. You can also load OpenIDM from a specified zip file (`"location" : "openidm.zip"`) or you can install a single jar file (`"location" : "openidm-system-2.1.jar"`).

  - `"includes" : "**/openidm-system-*.jar"` - the specified folder is scanned for jar files relating to the system startup. If the value of `"includes"` is `*.jar`, you must specifically exclude any jars in the bundle that you do not want to install, by setting the `"excludes"` property.

  - `"start-level" : 1` - specifies a start level for the jar files identified previously.

  - `"action" : "install.start"` - a period-separated list of actions to be taken on the jar files. Values can be one or more of `"install.start.update.uninstall"`.

  - `"config.properties"` - takes either a path to a configuration file (relative to the project location) or a list of configuration properties and their values. The list must be in the format `"string":"string"`, for example:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
"config.properties" :
  {
      "property" : "value"
  },
```

- `"system.properties"` - takes either a path to a `system.properties` file (relative to the project location) or a list of system properties and their values. The list must be in the format `"string":"string"`, for example:

```
"system.properties" :
  {
      "property" : "value"
  },
```

- `"boot.properties"` - takes either a path to a `boot.properties` file (relative to the project location) or a list of boot properties and their values.The list must be in the format `"string":object`, for example:

```
"boot.properties" :
  {
      "property" : true
  },
```

2.3. Obtaining Information About an OpenIDM Instance

OpenIDM includes a customizable information service that provides detailed information about a running OpenIDM instance. The information can be accessed over the REST interface, under the context `http://localhost:8080/openidm/info`.

By default, OpenIDM provides the following information:

- Basic information about the health of the system.

  This information can be accessed over REST at `http://localhost:8080/openidm/info/ping`. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/info/ping"

 {"state":"ACTIVE_READY","shortDesc":"OpenIDM ready"}
```

  The information is provided by the script `openidm/bin/defaults/script/info/ping.js`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

- Information about the current OpenIDM session.

  This information can be accessed over REST at `http://localhost:8080/openidm/info/login`. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/info/login"

 {
  "username":"openidm-admin",
  "userid":{
     "id":"openidm-admin",
     "component":"internal/user"
  }
 }
```

  The information is provided by the script `openidm/bin/defaults/script/info/login.js`.

You can extend or override the default information that is provided by creating your own script file and its corresponding configuration file in `openidm/conf/info-name.json`. Custom script files can be located anywhere, although a best practice is to place them in `openidm/script/info`. A sample customized script file for extending the default ping service is provided in `openidm/samples/infoservice/script/info/customping.js`. The corresponding configuration file is provided in `openidm/samples/infoservice/conf/info-customping.json`.

The configuration file has the following syntax:

```
{
    "infocontext" : "ping",
    "type" : "text/javascript",
    "file" : "script/info/customping.js"
}
```

The parameters in the configuration file are as follows:

- `"infocontext"` specifies the relative name of the info endpoint under the info context. The information can be accessed over REST at this endpoint, for example, setting `"infocontext"` to `"mycontext/myendpoint"` would make the information accessible over REST at `http://localhost:8080/openidm/info/mycontext/myendpoint`.
- `"type"` specifies the type of the information source. Currently, only Javascript is supported, so the type must be `"text/javascript"`.
- `"file"` specifies the path to the Javascript file, if you do not provide a `"source"` parameter.
- `"source"` specifies the actual Javascript, if you have not provided a `"file"` parameter.

Additional properties can be passed to the script in this configuration file (`openidm/samples/infoservice/conf/info- name.json`).

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Script files in `openidm/samples/infoservice/script/info/` have access to the following objects:

- `request` - the request details, including the method called and any parameters passed.
- `healthinfo` - the current health status of the system.
- `openidm` - access to the JSON resource API.
- Any additional properties that are defined in the configuration file (`openidm/samples/infoservice/conf/info-` *name*`.json`.)

2.4. Verifying the Health of an OpenIDM System

Due to the highly modular, configurable nature of OpenIDM, it is often difficult to assess whether a system has started up successfully, or whether the system is ready and stable after dynamic configuration changes have been made.

OpenIDM provides a configurable health check service that verifies that the required modules and services for an operational system are up and running. During system startup, OpenIDM checks that these modules and services are available and reports on whether any requirements for an operational system have not been met. If dynamic configuration changes are made, OpenIDM rechecks that the required modules and services are functioning so that system operation is monitored on an ongoing basis.

The health check service reports on the state of the OpenIDM system and outputs this state to the console and to the log files. The system can be in one of the following states:

`STARTING` - OpenIDM is starting up
`ACTIVE_READY` - all of the specified requirements have been met to consider the OpenIDM system ready
`ACTIVE_NOT_READY` - one or more of the specified requirements have not been met and the OpenIDM system is not considered ready
`STOPPING` - OpenIDM is shutting down

By default, OpenIDM checks the following modules and services:

Required Modules

```
"org.forgerock.openicf.framework.connector-framework"
"org.forgerock.openicf.framework.connector-framework-internal"
"org.forgerock.openicf.framework.connector-framework-osgi"
"org.forgerock.openidm.audit"
"org.forgerock.openidm.core"
"org.forgerock.openidm.enhanced-config"
"org.forgerock.openidm.external-email"
"org.forgerock.openidm.external-rest"
"org.forgerock.openidm.filter"
"org.forgerock.openidm.httpcontext"
"org.forgerock.openidm.infoservice"
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
"org.forgerock.openidm.policy"
"org.forgerock.openidm.provisioner"
"org.forgerock.openidm.provisioner-openicf"
"org.forgerock.openidm.repo"
"org.forgerock.openidm.restlet"
"org.forgerock.openidm.smartevent"
"org.forgerock.openidm.system"
"org.forgerock.openidm.ui"
"org.forgerock.openidm.util"
"org.forgerock.commons.org.forgerock.json.resource"
"org.forgerock.commons.org.forgerock.json.resource.restlet"
"org.forgerock.commons.org.forgerock.restlet"
"org.forgerock.commons.org.forgerock.util"
"org.forgerock.openidm.security-jetty"
"org.forgerock.openidm.jetty-fragment"
"org.forgerock.openidm.quartz-fragment"
"org.ops4j.pax.web.pax-web-extender-whiteboard"
"org.forgerock.openidm.scheduler"
"org.ops4j.pax.web.pax-web-jetty-bundle"
"org.forgerock.openidm.repo-jdbc"
"org.forgerock.openidm.repo-orientdb"
"org.forgerock.openidm.config"
"org.forgerock.openidm.crypto"
```

Required Services

```
"org.forgerock.openidm.config"
"org.forgerock.openidm.provisioner"
"org.forgerock.openidm.provisioner.openicf.connectorinfoprovider"
"org.forgerock.openidm.external.rest"
"org.forgerock.openidm.audit"
"org.forgerock.openidm.policy"
"org.forgerock.openidm.managed"
"org.forgerock.openidm.script"
"org.forgerock.openidm.crypto"
"org.forgerock.openidm.recon"
"org.forgerock.openidm.info"
"org.forgerock.openidm.router"
"org.forgerock.openidm.scheduler"
"org.forgerock.openidm.scope"
"org.forgerock.openidm.taskscanner"
```

You can replace this list, or add to it, by adding the following lines to the `openidm/conf/boot/boot.properties` file:

`"openidm.healthservice.reqbundles"` - overrides the default required bundles. Bundles are specified as a list of symbolic names, separated by commas.
`"openidm.healthservice.reqservices"` - overrides the default required services. Services are specified as a list of symolic names, separated by commas.
`"openidm.healthservice.additionalreqbundles"` - specifies required bundles (in addition to the default list). Bundles are specified as a list of symbolic names, separated by commas.
`"openidm.healthservice.additionalreqservices"` - specifies required services (in addition to the default list). Services are specified as a list of symbolic names, separated by commas.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

By default, OpenIDM gives the system ten seconds to start up all the required bundles and services, before the system readiness is assessed. Note that this is not the total start time, but the time required to complete the service startup after the framework has started. You can change this default by setting the value of the `servicestartmax` property (in miliseconds) in the `openidm/conf/boot/boot.properties` file. This example sets the startup time to five seconds.

```
openidm.healthservice.servicestartmax=5000
```

The health check service works in tandem with the scriptable information service. For more information see Section 2.3, "Obtaining Information About an OpenIDM Instance".

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 3

OpenIDM Command-Line Interface

OpenIDM includes a basic command-line interface that provides a number of utilities for managing the OpenIDM instance.

All of the utilities are subcommands of the `cli.sh` (UNIX) or `cli.bat` (Windows) scripts. To use the utilities, you can either run them as subcommands, or launch the **cli** script first, and then run the utility. For example, to run the **encrypt** utility on a UNIX system:

```
$ cd /path/to/openidm
$ ./cli.sh
Using boot properties at /openidm/conf/boot/boot.properties
openidm# encrypt ....
```

or

```
$ cd /path/to/openidm
$ ./cli.sh encrypt ...
```

By default, the command-line utilities run with the properties defined in `/path/to/openidm/conf/boot/boot.properties`.

The startup and shutdown scripts are not discussed in this chapter. For information about these scripts, see *Starting and Stopping OpenIDM*.

The following sections describe the subcommands and their use. Examples assume that you are running the commands on a UNIX system. For Windows systems, use **cli.bat** instead of **cli.sh**.

3.1. configexport

The **configexport** subcommand exports all configuration objects to a specified location, enabling you to reuse a system configuration in another environment. For example, you can test a configuration in a development environment, then export it and import it into a production environment. This subcommand also enables you to inspect the active configuration of an OpenIDM instance.

OpenIDM must be running when you execute this command.

Usage is as follows:

```
$ ./cli.sh configexport /export-location
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

For example:

```
$ ./cli.sh configexport /tmp/conf
```

Configuration objects are exported, as `.json` files, to the specified directory. Configuration files that are present in this directory are renamed as backup files, with a timestamp, for example, `audit.json.2012-12-19T12-00-28.bkp`, and are not overwritten. The following configuration objects are exported:

- The internal repository configuration (`repo.orientdb.json` or `repo.jdbc.json`)
- The log configuration (`audit.json`)
- The authentication configuration (`authentication.json`)
- The managed object configuration (`managed.json`)
- The connector configuration (`provisioner.openicf-*.json`)
- The router service configuration (`router.json`)
- The scheduler service configuration (`scheduler.json`)
- Any configured schedules (`schedule-*.json`)
- The synchronization mapping configuration (`sync.json`)
- If workflows are defined, the configuration of the workflow engine (`workflow.json`) and the workflow access configuration (`process-access.json`)
- Any configuration files related to the user interface (`ui-*.json`)
- The configuration of any custom endpoints (`endpoint-*.json`)
- The policy configuration (`policy.json`)

3.2. configimport

The **configimport** subcommand imports configuration objects from the specified directory, enabling you to reuse a system configuration from another environment. For example, you can test a configuration in a development environment, then export it and import it into a production environment.

The command updates the existing configuration from the *import-location* over the OpenIDM REST interface. By default, if configuration objects are present in the *import-location* and not in the existing configuration, these objects are added. If configuration objects are present in the existing location but not in the *import-location*, these objects are left untouched in the existing configuration.

If you include the `--replaceAll` parameter, the command wipes out the existing configuration and replaces it with the configuration in the *import-location*. Objects in the existing configuration that are not present in the *import-location* are deleted.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Usage is as follows:

```
$ ./cli.sh configimport [--replaceAll] /import-location
```

For example:

```
$ ./cli.sh configimport --replaceAll /tmp/conf
```

Configuration objects are imported, as `.json` files, from the specified directory to the `conf` directory. The configuration objects that are imported are outlined in the corresponding export command, described in the previous section.

3.3. configureconnector

The **configureconnector** subcommand generates a configuration for an OpenICF connector.

Usage is as follows:

```
$ ./cli.sh configureconnector connector-name
```

Select the type of connector that you want to configure. The following example configures a new XML connector.

```
$ ./cli.sh configureconnector myXmlConnector
Using boot properties at /openidm/conf/boot/boot.properties
Dec 11, 2012 10:35:37 AM org.restlet.ext.httpclient.HttpClientHelper start
INFO: Starting the HTTP client
0. CSV File Connector version 1.1.1.0
1. LDAP Connector version 1.1.1.0
2. org.forgerock.openicf.connectors.scriptedsql.ScriptedSQLConnector version 1.1.1.0
3. XML Connector version 1.1.1.0
4. Exit
Select [0..4]: 3
Edit the configuration file and run the command again. The configuration was
  saved to /openidm/temp/provisioner.openicf-myXmlConnector.json
```

The basic configuration is saved in a file named `/openidm/temp/provisioner.openicf-connector-name.json`. Edit the `configurationProperties` parameter in this file to complete the connector configuration. For an XML connector, you can use the schema definitions in sample 0 for an example configuration.

```
  "configurationProperties" : {
    "xmlFilePath" : "samples/sample0/data/resource-schema-1.xsd",
    "createFileIfNotExists" : false,
    "xsdFilePath" : "samples/sample0/data/resource-schema-extension.xsd",
    "xsdIcfFilePath" : "samples/sample0/data/xmlConnectorData.xml"
  },
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

For more information about the connector configuration properties, see *Configuring Connectors*.

When you have modified the file, run the **configureconnector** command again so that OpenIDM can pick up the new connector configuration.

```
$ ./cli.sh configureconnector myXmlConnector
Using boot properties at /openidm/conf/boot/boot.properties
Configuration was found and picked up from: /openidm/temp/provisioner.openicf-myXmlConnector.json
Dec 11, 2012 10:55:28 AM org.restlet.ext.httpclient.HttpClientHelper start
INFO: Starting the HTTP client
...
```

You can also configure connectors over the REST interface. For more information, see *Creating Default Connector Configurations*.

3.4. encrypt

The **encrypt** subcommand encrypts an input string, or JSON object, provided at the command line. This subcommand can be used to encrypt passwords, or other sensitive data, to be stored in the OpenIDM repository. The encrypted value is output to standard output and provides details of the cryptography key that is used to encrypt the data.

Usage is as follows:

```
$ ./cli.sh encrypt [-j] string
```

The `-j` option specifies that the string to be encrypted is a JSON object. If you do not enter the string as part of the command, the command prompts for the string to be encrypted. If you enter the string as part of the command, any special characters, for example quotation marks, must be escaped.

The following example encrypts a normal string value:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ ./cli.sh encrypt mypassword
Using boot properties at /openidm/conf/boot/boot.properties
Oct 23, 2012 2:00:03 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: Activating cryptography service of type: JCEKS provider:  location: security/keystore.jceks
Oct 23, 2012 2:00:03 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: Available cryptography key: openidm-sym-default
Oct 23, 2012 2:00:03 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: Available cryptography key: openidm-localhost
Oct 23, 2012 2:00:03 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: Available cryptography key: openidm-local-openidm-forgerock-org
Oct 23, 2012 2:00:03 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: CryptoService is initialized with 3 keys.
-----BEGIN ENCRYPTED VALUE-----
{
  "$crypto" : {
    "value" : {
      "iv" : "M2913T5ZADlC2ip2imeOyg==",
      "data" : "DZAAAM1nKjQM1qpLwh3BgA==",
      "cipher" : "AES/CBC/PKCS5Padding",
      "key" : "openidm-sym-default"
    },
    "type" : "x-simple-encryption"
  }
}
------END ENCRYPTED VALUE------
```

The following example encrypts a JSON object. The input string must be a valid JSON object.

```
$ ./cli.sh encrypt -j {\"password\":\"myPassw0rd\"}
Using boot properties at /openidm/conf/boot/boot.properties
Oct 23, 2012 2:00:03 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: Activating cryptography service of type: JCEKS provider:  location: security/keystore.jceks
Oct 23, 2012 2:00:03 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: Available cryptography key: openidm-sym-default
Oct 23, 2012 2:00:03 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: Available cryptography key: openidm-localhost
Oct 23, 2012 2:00:03 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: Available cryptography key: openidm-local-openidm-forgerock-org
Oct 23, 2012 2:00:03 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: CryptoService is initialized with 3 keys.
-----BEGIN ENCRYPTED VALUE-----
{
  "$crypto" : {
    "value" : {
      "iv" : "M2913T5ZADlC2ip2imeOyg==",
      "data" : "DZAAAM1nKjQM1qpLwh3BgA==",
      "cipher" : "AES/CBC/PKCS5Padding",
      "key" : "openidm-sym-default"
    },
    "type" : "x-simple-encryption"
  }
}
------END ENCRYPTED VALUE------
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The following example prompts for a JSON object to be encrypted. In this case, you need not escape the special characters.

```
$ ./cli.sh encrypt -j
Using boot properties at /openidm/conf/boot/boot.properties
Enter the Json value

> Press ctrl-D to finish input
Start data input:
{"password":"myPassw0rd"}
^D
Oct 23, 2012 2:37:56 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: Activating cryptography service of type: JCEKS provider:  location: security/keystore.jceks
Oct 23, 2012 2:37:56 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: Available cryptography key: openidm-sym-default
Oct 23, 2012 2:37:56 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: Available cryptography key: openidm-localhost
Oct 23, 2012 2:37:56 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: Available cryptography key: openidm-local-openidm-forgerock-org
Oct 23, 2012 2:37:56 PM org.forgerock.openidm.crypto.impl.CryptoServiceImpl activate
INFO: CryptoService is initialized with 3 keys.
-----BEGIN ENCRYPTED VALUE-----
{
  "$crypto" : {
    "value" : {
      "iv" : "6e0RK8/4F1EK5FzSZHwNYQ==",
      "data" : "gwHSdDTmzmUXeD6Gtfn6JFC8cAUiksiAGfvzTsdnAqQ=",
      "cipher" : "AES/CBC/PKCS5Padding",
      "key" : "openidm-sym-default"
    },
    "type" : "x-simple-encryption"
  }
}
------END ENCRYPTED VALUE------
```

3.5. keytool

The **keytool** subcommand exports or imports private key values.

The Java **keytool** command enables you to export and import public keys and certificates, but not private keys. The OpenIDM **keytool** subcommand provides this functionality.

Usage is as follows:

```
./cli.sh keytool [--export, --import] alias
```

For example, to export the default OpenIDM symmetric key, run the following command:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ ./cli.sh keytool --export openidm-sym-default
Using boot properties at /openidm/conf/boot/boot.properties
Use KeyStore from: /openidm/security/keystore.jceks
Please enter the password:
[OK] Secret key entry with algorithm AES
AES:606d80ae316be58e94439f91ad8ce1c0
```

The default keystore password is `changeit`. You should change this password after installation.

To import a new secret key named *my-new-key*, run the following command:

```
$ ./cli.sh keytool --import my-new-key
Using boot properties at /openidm/conf/boot/boot.properties
Use KeyStore from: /openidm/security/keystore.jceks
Please enter the password:
Enter the key:
AES:606d80ae316be58e94439f91ad8ce1c0
```

If a secret key of that name already exists, OpenIDM returns the following error:

```
"KeyStore contains a key with this alias"
```

3.6. validate

The **validate** subcommand validates all .json configuration files in the `openidm/conf/` directory.

Usage is as follows:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ ./cli.sh validate
Using boot properties at /openidm/conf/boot/boot.properties
...................................................................
[Validating] Load JSON configuration files from:
[Validating]  /openidm/conf
[Validating] audit.json ................................. SUCCESS
[Validating] authentication.json ........................ SUCCESS
[Validating] endpoint-getavailableuserstoassign.json ..... SUCCESS
[Validating] endpoint-getprocessesforuser.json ........... SUCCESS
[Validating] endpoint-gettasksview.json .................. SUCCESS
[Validating] endpoint-securityQA.json .................... SUCCESS
[Validating] endpoint-siteIdentification.json ............ SUCCESS
[Validating] endpoint-usernotifications.json ............. SUCCESS
[Validating] managed.json ................................ SUCCESS
[Validating] policy.json ................................. SUCCESS
[Validating] process-access.json ......................... SUCCESS
[Validating] provisioner.openicf-ldap.json ............... SUCCESS
[Validating] provisioner.openicf-xml.json ................ SUCCESS
[Validating] repo.orientdb.json .......................... SUCCESS
[Validating] router.json ................................. SUCCESS
[Validating] schedule-recon.json ......................... SUCCESS
[Validating] schedule-reconcile_systemXmlAccounts_managedUser.json  SUCCESS
[Validating] scheduler.json .............................. SUCCESS
[Validating] sync.json ................................... SUCCESS
[Validating] ui-configuration.json ....................... SUCCESS
[Validating] ui-countries.json ........................... SUCCESS
[Validating] ui-secquestions.json ........................ SUCCESS
[Validating] workflow.json ............................... SUCCESS
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 4
OpenIDM User Interface

OpenIDM provides a customizable, browser-based user interface. The default user interface enables administrative users to create, modify and delete user accounts. It provides role-based access to tasks based on BPMN2 workflows, and allows users to manage certain aspects of their own accounts, including configurable self-service registration.

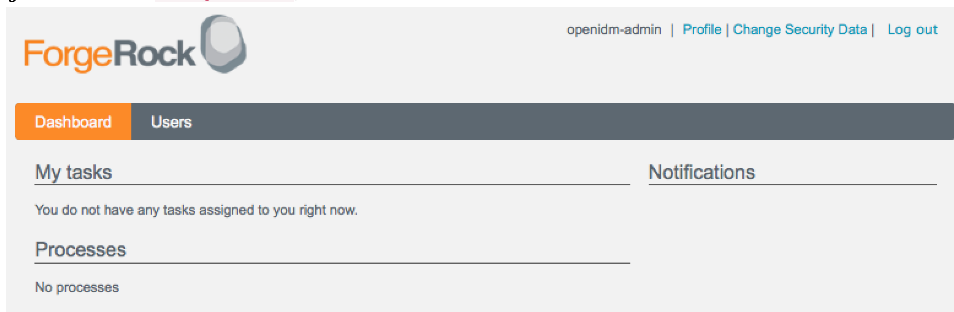
4.1. Overview of the Default User Interface

The default user interface is provided as a reference implementation that demonstrates the capabilities of the REST API. You can modify certain aspects of the default user interface according to the requirements of your deployment. Note, however, that the default user interface is still evolving and is not guaranteed to be compatible with the next OpenIDM release.

To access the user interface, install and start OpenIDM, then point your browser to http://localhost:8080/openidmui.

Log in as the default administrative user (Login: openidm-admin, Password: openidm-admin) or as an existing user in the repository. The display differs, depending on the role of the user that has logged in.

For an administrative user (role `openidm-admin`), two tabs are displayed - Dashboard and Users. The Dashboard tab lists any tasks assigned to the user, processes available to be invoked, and any notifications for that user. The Users tab provides an interface to manage user entries (OpenIDM managed objects under `managed/user`).



The `Profile` link enables the user to modify elements of his user data. The `Change Security Data` link enables the user to change his password and, optionally, to select a new security question.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

For a regular user (role `openidm-authorized`), the Users tab is not displayed - so regular users cannot manage user accounts, except for certain aspects of their own accounts.

4.2. Configuring the Default User Interface

The following sections outline the configurable aspects of the default user interface.

4.2.1. Enabling Self-Registration

Self-registration (the ability for new users to create their own accounts) is disabled by default. To enable self-registration, set `"selfRegistration"` to `true` in the `conf/ui-configuration.json` file.

```
{
    "configuration" : {
        "selfRegistration" : true,
...
```

With `"selfRegistration" : true`, the following capabilities are provided on the right-hand pane of the user interface:

Register my account
Reset my password

User objects created using self-registration automatically have the role `openidm-authorized`.

4.2.2. Configuring Security Questions

Security questions are disabled by default. To guard against unauthorized access, you can specify that users be prompted with security questions if they request a password reset. A default set of questions is provided, but you can add to these, or overwrite them. To enable security questions, set `"securityQuestions"` to `true` in the `conf/ui-configuration.json` file.

```
{
    "configuration" : {
        "securityQuestions" : true,
...
```

Specify the list of questions to be asked in the `conf/ui-secquestions.json` file.

Refresh your browser after this configuration change for the change to be picked up by the UI.

4.2.3. Enabling Site Identification

To ensure that users are entering their details onto the correct site, you can enable site identification. Site identification provides a preventative measure against phishing.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

With site identification enabled, users are presented with a range of images from which they can select. To enable site identification, set `"siteIdentification"` to `true` in the `conf/ui-configuration.json` file.

```
{
    "configuration" : {
        "siteIdentification" : true,
...
```

Refresh your browser after this configuration change for the change to be picked up by the UI.

A default list of four images is presented for site identification. The images are defined in the `siteImages` property in the `conf/ui-configuration.json` file:

```
"siteImages" : [
"images/passphrase/mail.png",
"images/passphrase/user.png",
"images/passphrase/report.png",
"images/passphrase/twitter.png"
],
...
```

The user selects one of these images, which is displayed on login. In addition, the user enters a Site Phrase, which is displayed beneath the site image on login. If either the site image, or site phrase is incorrect or absent when the user logs in, the user is aware that he is not logging in to the correct site.

You can change the default images, and include additional images, by placing image files in the `ui/extension/images` folder and modifying the `siteImages` property in the `ui-configuration.json` file to point to the new images. The following example assumes a file named `my-new-image.jpg`, located in `ui/extension/images`.

```
"siteImages" : [
"images/passphrase/mail.png",
"images/passphrase/user.png",
"images/passphrase/report.png",
"images/passphrase/twitter.png",
"images/my-new-image.jpg"
],
...
```

Note that the default image files are located in `ui/default/admin/public/images/passphrase`.

4.2.4. Configuring the Country List

The default user profile includes the ability to specify the user's country and state or province. To specify the countries that appear in this drop down list, and their associated states or provinces, edit the `conf/ui-countries.json` file. For example, to add Norway to the list of countries, you would add the following to the `conf/ui-countries.json` file:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
        {
            "key" : "norway",
            "value" : "Norway",
            "states" : [
                {
                    "key" : "akershus",
                    "value" : "Akershus"
                },
                {
                    "key" : "aust-agder",
                    "value" : "Aust-Agder"
                },
                {
                    "key" : "buskerud",
                    "value" : "Buskerud"
                },
...
```

Refresh your browser after this configuration change for the change to be picked up by the UI.

4.3. Managing User Accounts With the User Interface

Only administrative users (with the role `openidm-admin`) can add, modify, and delete user accounts. Regular users can modify certain aspects of their own accounts.

Procedure 4.1. To Add a User Account

1. Log into the user interface as an administrative user.
2. Select the Users tab.
3. Click Add User.
4. Complete the fields on the Create new account page.

   Most of these fields are self-explanatory. Be aware that the user interface is subject to policy validation, as described in *Using Policies to Validate Data*. So, for example, the Email address must be of valid email address format, and the Password must comply with the password validation settings that are indicated in the panel to the right.

   The Admin Role field reflects the roles that are defined in the `ui-configuration.json` file. The roles are mapped as follows:

   ```
   "roles" : {
       "openidm-admin" : "Administrator",
       "openidm-authorized" : "User",
       "tasks-manager" : "Tasks Manager"
   },
   ```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

By default, a user can be assigned more than one role. Only users with the `tasks-manager` role can assign tasks to any candidate user for that task.

Procedure 4.2. To Update a User Account

1. Log into the user interface as an administrative user.
2. Select the Users tab.
3. Click the Username of the user that you want to update.
4. On the user's profile page, modify the fields you want to change and click Update.

   The user account is updated in the internal repository.

Procedure 4.3. To Reset a User's Password

Users can change their own passwords by following the Change Security Data link in their profiles. This process requires that users know their existing passwords.

In a situation where a user forgets his password, an administrator can reset the password of that user without knowing the user's existing password.

1. Follow steps 1-3 in Procedure 4.2, "To Update a User Account".
2. On the user's profile page, click Change password.
3. Enter a new password that conforms to the password policy and click Update.

   The user password is updated in the repository.

Procedure 4.4. To Delete a User Account

1. Log into the user interface as an administrative user.
2. Select the Users tab.
3. Click the Username of the user that you want to delete.
4. On the user's profile page, click Delete.
5. Click OK to confirm the deletion.

   The user is deleted from the internal repository.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

4.4. Managing Workflows From the User Interface

The user interface is integrated with the embedded Activiti worfklow engine, enabling users to interact with workflows. Available workflows are displayed under the Processes item on the Dashboard. In order for a workflow to be displayed here, the workflow definition file must be present in the `openidm/workflow` directory.

A sample workflow integration with the user interface is provided in `openidm/samples/workflow`, and documented in *Sample Workflow - Provisioning User Accounts*. Follow the steps in that sample for an understanding of how the workflow integration works.

Access to workflows is based on OpenIDM roles, and is configured in the file `conf/process-access.json`. By default all users with the role `openidm-authorized` or `openidm-admin` can invoke any available workflow. The default `process-access.json` file is as follows:

```
{
    "workflowAccess" : [
        {
            "propertiesCheck" : {
                "property" : "_id",
                "matches" : ".*",
                "requiresRole" : "openidm-authorized"
            }
        },
        {
            "propertiesCheck" : {
                "property" : "_id",
                "matches" : ".*",
                "requiresRole" : "openidm-admin"
            }
        }
    ]
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 5

Configuring OpenIDM

OpenIDM configuration is split between `.properties` and container configuration files, and also dynamic configuration objects. The majority of OpenIDM configuration files are stored under `openidm/conf/`, as described in the appendix listing the *File Layout*.

OpenIDM stores configuration objects in its internal repository. You can manage the configuration by using either the REST access to the configuration objects, or by using the JSON file based views.

5.1. OpenIDM Configuration Objects

OpenIDM exposes internal configuration objects in JSON format. Configuration elements can be either single instance or multiple instance for an OpenIDM installation.

Single Instance Configuration Objects

Single instance configuration objects correspond to services that have at most one instance per installation.

JSON file views of these configuration objects are named `object-name.json`.

- The `audit` configuration specifies how audit events are logged.
- The `authentication` configuration controls REST access.
- The `endpoint` configuration controls any custom REST endpoints.
- The `managed` configuration defines managed objects and their schemas.
- The `policy` configuration defines the policy validation service.
- The `process access` configuration defines access to any configured workflows.
- The `repo.repo-type` configuration such as `repo.orientdb` or `repo.jdbc` configures the internal repository.
- The `router` configuration specifies filters to apply for specific operations.
- The `sync` configuration defines the mappings that OpenIDM uses when synchronizing and reconciling managed objects.
- The `ui` configuration defines the configurable aspects of the default user interface.
- The `workflow` configuration defines the configuration of the workflow engine.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Multiple Instance Configuration Objects

Multiple instance configuration objects correspond to services that can have many instances per installation. Configuration objects are named `objectname/instancename`. For example, `provisioner.openicf/xml`.

JSON file views of these configuration objects are named `objectname-instancename.json`. For example, `provisioner.openicf-xml.json.`

- Multiple `schedule` configurations can run reconciliations and other tasks on different schedules.
- Multiple `provisioner.openicf` configurations correspond to the resources connected to OpenIDM.

5.2. Changing the Default Configuration

When you change OpenIDM's configuration objects, take the following points into account.

- OpenIDM's authoritative configuration source is the internal repository. JSON files provide a view of the configuration objects, but do not represent the authoritative source.

  OpenIDM updates JSON files after making configuration changes, whether those changes are made through REST access to configuration objects, or through edits to the JSON files.

- OpenIDM recognizes changes to JSON files when it is running. OpenIDM *must* be running when you delete configuration objects, even if you do so by editing the JSON files.
- Avoid editing configuration objects directly in the internal repository. Rather edit the configuration over the REST API, or in the configuration JSON files to ensure consistent behavior and that operations are logged.
- OpenIDM stores its configuration in the internal database by default. If you remove an OpenIDM instance and do not specifically drop the repository, the configuration remains in effect for a new OpenIDM instance that uses that repository. For testing or evaluation purposes, you can disable this *persistent configuration* in the `conf/system.properties` file by uncommenting the following line:

  ```
  # openidm.config.repo.enabled=false
  ```

  Disabling persistent configuration means that OpenIDM will store its configuration in memory only. You should not disable persistent configuration in a production environment.

5.3. Configuring an OpenIDM System for Production

Out of the box, OpenIDM is configured to make it easy to install and evaluate. Specific configuration changes are required before you deploy OpenIDM in a production environment.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

5.3.1. Configuring a Production Repository

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 you use OpenIDM in production, you must replace OrientDB with a supported repository.

For more information, see *Installing a Repository for Production* in the *Installation Guide* in the *Installation Guide*.

5.3.2. Disabling Automatic Configuration Updates

By default, OpenIDM polls the JSON files in the `conf` directory periodically for any changes to the configuration. In a production system, it is recommended that you disable automatic polling for updates to prevent untested configuration changes from disrupting your identity service.

To disable automatic polling for configuration changes, edit the `conf/system.properties` file by uncommenting the following line:

```
                # openidm.fileinstall.enabled=false
```

Before you disable automatic polling, you must have started the OpenIDM instance at least once to ensure that the configuration has been loaded into the database.

Note that scripts are loaded each time the configuration calls the script. Modifications to scripts are therefore not applied dynamically. If you modify a script, you must either modify the configuration that calls the script, or restart the component that uses the modified script. You do not need to restart OpenIDM for script modifications to take effect.

5.3.3. Disabling the File-Based Configuration View

To control configuration changes to the OpenIDM system, you disable the file-based configuration view and have OpenIDM read its configuration only from the repository. To disable the file-based configuration view, edit the `conf/system.properties` file to uncomment the following line: `# openidm.fileinstall.enabled=false`.

5.4. Configuring OpenIDM Over REST

OpenIDM exposes configuration objects under the `/openidm/config` context.

You can list the configuration on the local host by performing a GET `http://localhost:8080/openidm/config`. The following example shows the default configuration for an OpenIDM instance started with Sample 1.

```
$ curl --request GET
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 http://localhost:8080/openidm/config
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
    "configurations": [
        {
            "_id": "endpoint/getprocessesforuser",
            "pid": "endpoint.788f364e-d870-4f46-982a-793525fff6f0",
            "factoryPid": "endpoint"
        },
        {
            "_id": "provisioner.openicf/xml",
            "pid": "provisioner.openicf.90b18af9-fe27-45a2-a4ae-1056c04a4d31",
            "factoryPid": "provisioner.openicf"
        },
        {
            "_id": "ui/configuration",
            "pid": "ui.36bb2bf4-8e19-43d2-9df2-a0553ffac590",
            "factoryPid": "ui"
        },
        {
            "_id": "managed",
            "pid": "managed",
            "factoryPid": null
        },
        {
            "_id": "sync",
            "pid": "sync",
            "factoryPid": null
        },
        {
            "_id": "router",
            "pid": "router",
            "factoryPid": null
        },
        {
            "_id": "process/access",
            "pid": "process.44743c97-a01b-4562-85ad-8a2c9b89155a",
            "factoryPid": "process"
        },
        {
            "_id": "endpoint/siteIdentification",
            "pid": "endpoint.ef05a7f3-a420-4fbb-998c-02d283cae4d1",
            "factoryPid": "endpoint"
        },
        {
            "_id": "endpoint/securityQA",
            "pid": "endpoint.e2d87637-c918-4056-99a1-20f25c897066",
            "factoryPid": "endpoint"
        },
        {
            "_id": "scheduler",
            "pid": "scheduler",
            "factoryPid": null
        },
        {
            "_id": "ui/countries",
            "pid": "ui.acde0f4c-808f-45fb-9627-d7d2ca702e7c",
            "factoryPid": "ui"
        },
        {
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
            "_id": "org.apache.felix.fileinstall/openidm",
            "pid": "org.apache.felix.fileinstall.2dedea63-4592-4074-a709-ffa70f1e841d",
            "factoryPid": "org.apache.felix.fileinstall"
        },
        {
            "_id": "schedule/reconcile_systemXmlAccounts_managedUser",
            "pid": "schedule.f53b235a-862e-4e18-a3cf-10ae3cbabc1e",
            "factoryPid": "schedule"
        },
        {
            "_id": "workflow",
            "pid": "workflow",
            "factoryPid": null
        },
        {
            "_id": "endpoint/getavailableuserstoassign",
            "pid": "endpoint.d19da94f-bae3-4101-922c-fe47ea8616d2",
            "factoryPid": "endpoint"
        },
        {
            "_id": "repo.orientdb",
            "pid": "repo.orientdb",
            "factoryPid": null
        },
        {
            "_id": "audit",
            "pid": "audit",
            "factoryPid": null
        },
        {
            "_id": "endpoint/gettasksview",
            "pid": "endpoint.edcc1ff8-a7ba-4c46-8258-bf5216e85192",
            "factoryPid": "endpoint"
        },
        {
            "_id": "ui/secquestions",
            "pid": "ui.649e2c65-0cc7-4a0d-a6b1-95f4c5168bdc",
            "factoryPid": "ui"
        },
        {
            "_id": "org.apache.felix.fileinstall/activiti",
            "pid": "org.apache.felix.fileinstall.a0ba2f7d-bdb9-43b5-b84e-0e8feee6be72",
            "factoryPid": "org.apache.felix.fileinstall"
        },
        {
            "_id": "policy",
            "pid": "policy",
            "factoryPid": null
        },
        {
            "_id": "endpoint/usernotifications",
            "pid": "endpoint.e96d5319-6260-41db-af76-bd4e692b792d",
            "factoryPid": "endpoint"
        },
        {
            "_id": "org.apache.felix.fileinstall/ui",
            "pid": "org.apache.felix.fileinstall.89f8c6dd-f54e-46a4-bfda-1e76ac044c33",
            "factoryPid": "org.apache.felix.fileinstall"
        },
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
        {
            "_id": "authentication",
            "pid": "authentication",
            "factoryPid": null
        }
    ]
}
```

Single instance configuration objects are located under `openidm/config/object-name`. The following example shows the default `audit` configuration.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 http://localhost:8080/openidm/config/audit

{
    "eventTypes": {
        "activity": {
            "filter": {
                "actions": [
                    "create",
                    "update",
                    "delete",
                    "patch",
                    "action"
                ]
            }
        },
        "recon": {}
    },
    "logTo": [
        {
            "logType": "csv",
            "location": "audit",
            "recordDelimiter": ";"
        },
        {
            "logType": "repository"
        }
    ]
}
```

Multiple instance configuration objects are found under `openidm/config/object-name/instance-name`. The following example shows the configuration for the XML connector provisioner.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 http://localhost:8080/openidm/config/provisioner.openicf/xml

{
    "name": "xmlfile",
    "connectorRef": {
        "bundleName":
            "org.forgerock.openicf.connectors.file.openicf-xml-connector",
        "bundleVersion": "1.1.1.0",
        "connectorName": "com.forgerock.openicf.xml.XMLConnector"
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
    },
    "producerBufferSize": 100,
    "connectorPoolingSupported": true,
    "poolConfigOption": {
        "maxObjects": 10,
        "maxIdle": 10,
        "maxWait": 150000,
        "minEvictableIdleTimeMillis": 120000,
        "minIdle": 1
    },
    "operationTimeout": {
        "CREATE": -1,
        "TEST": -1,
        "AUTHENTICATE": -1,
        "SEARCH": -1,
        "VALIDATE": -1,
        "GET": -1,
        "UPDATE": -1,
        "DELETE": -1,
        "SCRIPT_ON_CONNECTOR": -1,
        "SCRIPT_ON_RESOURCE": -1,
        "SYNC": -1,
        "SCHEMA": -1
    },
    "configurationProperties": {
        "xsdIcfFilePath": "samples/sample1/data/resource-schema-1.xsd",
        "xsdFilePath": "samples/sample1/data/resource-schema-extension.xsd",
        "xmlFilePath": "samples/sample1/data/xmlConnectorData.xml"
    },
    "objectTypes": {
        "account": {
            "$schema": "http://json-schema.org/draft-03/schema",
            "id": "__ACCOUNT__",
            "type": "object",
            "nativeType": "__ACCOUNT__",
            "properties": {
                "description": {
                    "type": "string",
                    "nativeName": "__DESCRIPTION__",
                    "nativeType": "string"
                },
                "firstname": {
                    "type": "string",
                    "nativeName": "firstname",
                    "nativeType": "string"
                },
                "email": {
                    "type": "string",
                    "nativeName": "email",
                    "nativeType": "string"
                },
                "__UID__": {
                    "type": "string",
                    "nativeName": "__UID__"
                },
                "password": {
                    "type": "string",
                    "required": false,
                    "nativeName": "__PASSWORD__",
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
                    "nativeType": "JAVA_TYPE_GUARDEDSTRING",
                    "flags": [
                        "NOT_READABLE",
                        "NOT_RETURNED_BY_DEFAULT"
                    ]
                },
                "name": {
                    "type": "string",
                    "required": true,
                    "nativeName": "__NAME__",
                    "nativeType": "string"
                },
                "lastname": {
                    "type": "string",
                    "required": true,
                    "nativeName": "lastname",
                    "nativeType": "string"
                }
            }
        }
    },
    "operationOptions": {}
}
```

You can change the configuration over REST by using an HTTP PUT request to modify the required configuration object. The following example modifies the `router.json` file to remove all filters, effectively bypassing any policy validation.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request PUT
 --data '{
          "filters" : [
             {
                "onRequest" : {
                   "type" : "text/javascript",
                   "file" : "bin/defaults/script/router-authz.js"
                }
              }
           ]
           }'
 "http://localhost:8080/openidm/config/router"
```

See the *REST API Reference* appendix for additional details and examples using REST access to update and patch objects.

5.5. Using Property Value Substitution in the Configuration

In an environment where you have more than one OpenIDM instance, you might require a configuration that is similar, but not identical, across the different OpenIDM hosts. OpenIDM supports variable replacement in its configuration which means that you can modify the effective configuration according to the requirements of a specific environment or OpenIDM instance.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Property substitution enables you to achieve the following:

- Define a configuration that is specific to a single OpenIDM instance, for example, setting the location of the keystore on a particular host.
- Define a configuration whose parameters vary between different environments, for example, the URLs and passwords for test, development, and production environments.
- Disable certain capabilities on specific nodes. For example, you might want to disable the workflow engine on specific instances.

When OpenIDM starts up, it combines the system configuration, which might contain specific environment variables, with the defined OpenIDM configuration properties. This combination makes up the effective configuration for that OpenIDM instance. By varying the environment properties, you can change specific configuration items that vary between OpenIDM instances or environments.

Property references are contained within the construct `&{ }`. When such references are found, OpenIDM replaces them with the appropriate property value, defined in the `boot.properties` file.

Example 5.1.

The following example defines two separate OpenIDM environments - a development environment and a production environment. You can specify the environment at startup time and, depending on the environment, the database URL is set accordingly.

The environments are defined by adding the following lines to the `conf/boot.properties` file:

```
PROD.location=production
DEV.location=development
```

The database URL is then specified as follows in the `repo.orientdb.json` file:

```
{
"dbUrl" : "local:./db/&{&{environment}.location}-openidm",
"user" : "admin",
"poolMinSize" : 5,
"poolMaxSize" : 20,
...
}
```

The effective database URL is determined by setting the `OPENIDM_OPTS` environment variable when you start OpenIDM. To use the production environment, start OpenIDM as follows:

```
$ export OPENIDM_OPTS="-Xmx1024m -Denvironment=PROD"
$ ./startup.sh
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

To use the development environment, start OpenIDM as follows:

```
$ export OPENIDM_OPTS="-Xmx1024m -Denvironment=DEV"
$ ./startup.sh
```

5.5.1. Using Property Value Substitution With System Properties

You can use property value substitution in conjunction with the system properties, to modify the configuration according to the system on which the OpenIDM instance runs.

Example 5.2.

The following example modifies the `audit.json` file so that the log file is written to the user's directory. The `user.home` property is a default Java System property.

```
{
    "logTo" : [
        {
            "logType" : "csv",
            "location" : "&{user.home}/audit",
            "recordDelimiter" : ";"
        }
    ]
}
```

You can define *nested* properties (that is a property definition within another property definition) and you can combine system properties and boot properties.

Example 5.3.

The following example uses the `user.country` property, a default Java System property. The example defines specific LDAP ports, depending on the country (identified by the country code) in the `boot.properties` file. The value of the LDAP port (set in the `provisioner.openicf-ldap.json` file) depends on the value of the `user.country` System property.

The port numbers are defined in the `boot.properties` file as follows:

```
openidm.NO.ldap.port=2389
openidm.EN.ldap.port=3389
openidm.US.ldap.port=1389
```

The following extract from the `provisioner.openicf-ldap.json` file shows how the value of the LDAP port is eventually determined, based on the System property:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
"configurationProperties" :
{
   "credentials" : "Passw0rd",
   "port" : "&{openidm.&{user.country}.ldap.port}",
   "principal" : "cn=Directory Manager",
   "baseContexts" :
      [
         "dc=example,dc=com"
      ],
   "host" : "localhost"
}
```

5.5.2. Limitations of Property Value Substitution

Note the following limitations when you use property value substitution:

- You cannot reference complex objects or properties with syntaxes other than String. Property values are resolved from the `boot.properties` file or from the System properties and the value of these properties is always in String format.

  Property substitution of boolean values is currently only supported in stringified format, that is, resulting in `"true"` or `"false"`.

- Substitution of encrypted property values is currently not supported.

5.6. Adding Custom Endpoints

You can customize OpenIDM to meet the specific requirements of your deployment by adding your own RESTful endpoints. Endpoints are configured in files named `conf/endpoint-name.json`, where *name* generally describes the purpose of the endpoint.

A sample custom endpoint configuration is provided at `openidm/samples/customendpoint`. The sample includes two files:

`conf/endpoint-echo.json`, which provides the configuration for the endpoint.
`script/echo.js`, which is launched when the endpoint is accessed.

The structure of an endpoint configuration file is as follows:

```
{
  "context" : "endpoint/echo",
  "type" : "text/javascript",
  "file" : "script/echo.js"
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`"context"`

The URL context under which the endpoint is registered. Currently this *must* be under `endpoint/`. An endpoint registered under the context `endpoint/echo` is addressable over REST at `http://localhost:8080/openidm/endpoint/echo` and with the internal resource API, for example `openidm.read("endpoint/echo")`.

`"type"`

The type of implementation. Currently only `"text/javascript"` is supported.

`"source"` or `"file`

The actual script, inline, or a pointer to the file that contains the script. The sample script, (`samples/customendpoint/script/echo.js`) simply returns the HTTP request when a request is made on that endpoint.

The endpoint script has a `request` variable available in its scope. The request structure carries all the information about the request, and includes the following properties:

`id`

The local ID, without the `endpoint/` prefix, for example, `echo`.

`method`

The requested operation, that is, `create`, `read`, `update`, `delete`, `patch`, `query` or `action`.

`params`

The parameters that are passed in. For example, for an HTTP GET with `?param=x`, the request contains `"params":{"param":"x"}`.

`parent`

Provides the context for the invocation, including headers and security.

Note that the interface for this context is still evolving and may change in a future release.

A script implementation should check and reject requests for methods that it does not support. For example, the following implementation supports only the `read` method:

```
if (request.method == "read") {
    ...
} else {
    throw "Unsupported operation: " + request.method;
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The final statement in the script is the return value. Unlike for functions, at this global scope there is no `return` keyword. In the following example, the value of the last statement (`x`) is returned.

```
var x = "Sample return"
functioncall();
x
```

The following example uses the sample provided in `openidm/samples/customendpoint` and shows the complete request structure, which is returned by the query.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/endpoint/echo?param=x"
{
  "type": "resource",
  "uuid": "21c5ddc6-a66e-464e-9fa4-9b777505799e",
  "params": {
    "param": "x"
  },
  "method": "query",
  "parent": {
    "path": "/openidm/endpoint/echo",
    "headers": {
      "Accept": "*/*",
      "User-Agent": "curl/7.21.4 ... OpenSSL/0.9.8r zlib/1.2.5",
      "Authorization": "Basic b3BlbmlkbS1hZG1pbjpvcGVuaWRtLWFkbWlu",
      "Host": "localhost:8080"
    },
    "query": {
      "param": "x"
    },
    "method": "GET",
    "parent": {
      "uuid": "bec97cbf-8618-42f8-a841-9f5f112538e9",
      "parent": null,
      "type": "root"
    },
    "type": "http",
    "uuid": "7fb3e0d9-5f56-4b15-b710-28f2147cf4b4",
    "security": {
      "openidm-roles": [
        "openidm-admin",
        "openidm-authorized"
      ],
      "username": "openidm-admin",
      "userid": {
        "component": "internal/user",
        "id": "openidm-admin"
      }
    }
  },
  "id": "echo"
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

You must protect access to any custom endpoints by configuring the appropriate authorization for those contexts. For more information, see the *Authorization* section.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 6 Accessing Data Objects

OpenIDM supports a variety of objects that can be addressed via a URL or URI. You can access data objects by using scripts (through the Resource API) or by using direct HTTP calls (through the REST API).

The following sections describe these two methods of accessing data objects, and provide information on constructing and calling data queries.

6.1. Accessing Data Objects by Using Scripts

OpenIDM's uniform programming model means that all objects are queried and manipulated in the same way, using the Resource API. The URL or URI that is used to identify the target object for an operation depends on the object type. For an explanation of object types, see the *Data Models and Objects Reference*. For more information about scripts and the objects available to scripts, see the *Scripting Reference*.

You can use the Resource API to obtain managed objects, configuration objects, and repository objects, as follows:

```
val = openidm.read("managed/organization/mysampleorg")

val = openidm.read("config/custom/mylookuptable")

val = openidm.read("repo/custom/mylookuptable")
```

For information about constructing an object ID, see *URI Scheme* in the *REST API Reference*.

You can update entire objects with the `update()` function, as follows.

```
openidm.update("managed/organization/mysampleorg", mymap)

openidm.update("config/custom/mylookuptable", mymap)

openidm.update("repo/custom/mylookuptable", mymap)
```

For managed objects, you can partially update an object with the `patch()` function.

```
openidm.patch("managed/organization/mysampleorg", rev, value)
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The `create()`, `delete()`, and `query()` functions work the same way.

6.2. Accessing Data Objects by Using the REST API

OpenIDM provides RESTful access to data objects via a REST API. To access objects 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.

For a comprehensive overview of the REST API, see the *REST API Reference* appendix.

To obtain a managed object through the REST API, depending on your security settings and authentication configuration, perform an HTTP GET on the corresponding URL, for example `https://localhost:8443/openidm/managed/organization/mysampleorg`.

By default, the HTTP GET returns a JSON representation of the object.

6.3. Defining and Calling Queries

OpenIDM supports an advanced query model that enables you to define queries, and to call them over the REST or Resource API.

6.3.1. Parameterized Queries

Managed objects in the supported OpenIDM repositories can be accessed using a parameterized query mechanism. Parameterized queries on repositories are defined in the repository configuration (`repo.*.json`) and are called by their `_queryId`.

Parameterized queries provide security and portability for the query call signature, regardless of the back-end implementation. Queries that are exposed over the REST interface *must* be parameterized queries to guard against injection attacks and other misuse. Queries on the officially supported repositories have been reviewed and hardened against injection attacks.

For system objects, support for parameterized queries is restricted to `_queryId=query-all-ids`. There is currently no support for user-defined parameterized queries on system objects. Typically, parameterized queries on system objects are not called directly over the REST interface, but are issued from internal calls, such as correlation queries.

A typical query definition is as follows:

```
"query-all-ids" : "select _openidm_id from ${unquoted:_resource}"
```

To call this query, you would reference its ID, as follows:

```
?_queryId=query-all-ids
```

The following example calls `query-all-ids` over the REST interface:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

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

6.3.2. Native Query Expressions

Native query expressions are supported for all managed objects and system objects, and can be called directly over the REST interface, rather than being defined in the repository configuration.

Native queries are intended specifically for internal callers, such as custom scripts, in situations where the parameterized query facility is insufficient. For example, native queries are useful if the query needs to be generated dynamically.

The query expression is specific to the target resource. For repositories, queries use the native language of the underlying data store. For system objects that are backed by OpenICF connectors, queries use the applicable query language of the system resource.

Native queries on the repository are made using the `_queryExpression` keyword. For example:

```
$ curl
                --header "X-OpenIDM-Username: openidm-admin"
                --header "X-OpenIDM-Password: openidm-admin"
                "http://localhost:8080/openidm/managed/user?_queryExpression=select+*+from+managedobjects"
```

Unlike parameterized queries, native queries are not portable and do not guard against injection attacks. Such query expressions should therefore not be used or made accessible over the REST interface or over HTTP, other than for development, and should be used only via the internal Resource API. For more information, see the section on *Protecting Sensitive REST Interface URLs*.

If you really need to expose native queries over HTTP, in a selective manner, you can design a custom endpoint to wrap such access.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 7
Using Policies to Validate Data

OpenIDM provides an extensible policy service that enables you to apply specific validation requirements to various components and properties. The policy service provides a REST interface for reading policy requirements and validating the properties of components against configured policies. Objects and properties are validated automatically when they are created, updated, or patched. Policies can be applied to user passwords, but also to any kind of managed object.

The policy service enables you to do the following:

- Read the configured policy requirements of a specific component.
- Read the configured policy requirements of all components.
- Validate a component object against the configured policies.
- Validate the properties of a component against the configured policies.

A default policy applies to all managed objects. You can configure the default policy to suit your requirements, or you can extend the policy service by supplying your own scripted policies.

7.1. Configuring the Default Policy

The default policy is configured in two files:

- A policy script file (`openidm/bin/defaults/script/policy.js`) which defines each policy and specifies how policy validation is performed.
- A policy configuration file (`openidm/conf/policy.json`) which specifies which policies are applicable to each resource.

7.1.1. Policy Script File

The policy script file defines policy configuration in two parts:

- A policy configuration object, which defines each element of the policy.
- A policy implementation function, which describes the requirements that are enforced by that policy.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Together, the configuration object and the implementation function determine whether an object is valid in terms of the policy. The following extract from the policy script file configures a policy that specifies that the value of a property must contain a certain number of capital letters.

```
...
            {   "policyId" : "at-least-X-capitals",
                "clientValidation": true,
                "policyExec" : "atLeastXCapitalLetters",
                "policyRequirements" : ["AT_LEAST_X_CAPITAL_LETTERS"]
            },
...

function atLeastXCapitalLetters(fullObject, value, params, property) {
    var reg = /[(A-Z)]/g;
    if (typeof value !== "string" || !value.length || value.match(reg)
        === null || value.match(reg).length < params.numCaps) {
        return [ {
                   "policyRequirement" : "AT_LEAST_X_CAPITAL_LETTERS",
                   "params" : {
                     "numCaps": params.numCaps
                   }
                  }
               ];
    }
    return [];
}
...
```

To enforce user passwords that contain at least one capital letter, the previous policy ID is applied to the appropriate resource and the required number of capital letters is defined in the policy configuration file, as described in Section 7.1.2, "Policy Configuration File".

7.1.1.1. Policy Configuration Object

Each element of the policy is defined in a policy configuration object. The structure of a policy configuration object is as follows:

```
            {   "policyId" : "minimum-length",
                "clientValidation": true,
                "policyExec" : "propertyMinLength",
                "policyRequirements" : ["MIN_LENGTH"]
            }
```

`"policyId"` - a unique ID that enables the policy to be referenced by component objects.
`"clientValidation"` - indicates whether the policy decision can be made on the client. When `"clientValidation": true`, the source code for the policy decision function is returned when the client requests the requirements for a property.
`"policyExec"` - the name of the function that contains the policy implementation. For more information, see Section 7.1.1.2, "Policy Implementation Function".

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`"policyRequirements"` - an array containing the policy requirement ID of each requirement that is associated with the policy. Typically, a policy will validate only one requirement, but it can validate more than one.

7.1.1.2. Policy Implementation Function

Each policy ID has a corresponding policy implementation function that performs the validation. Functions take the following form:

```
function <name>(fullObject, value, params, propName) {
 <implementation_logic>
}
```

`fullObject` is the full resource object that is supplied with the request.
`value` is the value of the property that is being validated.
`params` refers to the `"params"` array that is specified in the property's policy configuration.
`propName` is the name of the property that is being validated.

The following example shows the implementation function for the `"required"` policy.

```
function required(fullObject, value, params, propName) {
    if (value === undefined) {
        return [ { "policyRequirement" : "REQUIRED" } : "REQUIRED" } ];
    }
    return [];
}
```

7.1.2. Policy Configuration File

The policy configuration file includes a pointer to the policy script, and the configured policies for each component resource. The following extract of the default policy configuration file shows how the `at-least-X-capitals` policy is applied to user passwords, with a default value of `1`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
    "type" : "text/javascript",
    "file" : "bin/defaults/script/policy.js",
    "resources" : [
        {
            "resource" : "managed/user/*",
            "properties" : [
...
                {
                    "name" : "password",
                    "policies" : [
                        {
                            "policyId" : "required"
                        },
                        {
                            "policyId" : "not-empty"
                        },
                        {
                            "policyId" : "at-least-X-capitals",
                            "params" : {
                                "numCaps" : 1
                            }
                        },
                ...
            }
        ]
}
```

The configuration file includes the following properties:

- `"type"` - specifies the type of policy service. Currently, only `"text/javascript"` is supported.
- `"file"` - provides the path to the policy script file, relative to the OpenIDM installation directory.
- `"resources"` provides an array of resource objects, in JSON format, that are subject to the policy service. Resource objects are identified by the `"resource"` parameter, which indicates the URI and supports wildcard syntax. For example, `"managed/user/*"` indicates that the policy applies to all objects under `/managed/user`. Each resource has the following properties:

  `"name"` - the name of the property to which the policy is applied.
  `"policyID"` - the ID of the policy that is applied to that property.
  `"params"` - any specific parameters that apply to that policy ID.

You can specify that a particular policy does not apply to users with specific OpenIDM roles by setting the `"exceptRoles"` parameter of the policy ID. For example, the following extract from `policy.json` specifies that the reauthorization required policy definition does not apply to users with roles `openidm-admin`, or `opendim-reg`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
...
    {
        "policyId" : "re-auth-required",
        "params" : {
            "exceptRoles" : [
                "openidm-admin",
                "openidm-reg"
            ]
        }
    }
...
```

7.2. Extending the Policy Service

You can extend the policy service by adding your own scripted policies in `openidm/script` and referencing them in the policy configuration file (`conf/policy.json`). Avoid manipulating the default policy script file (in `bin/defaults/script`) as doing so might result in interoperability issues in a future release. To reference additional policy scripts, set the `"additionalFiles"` property in `conf/policy.json`.

The following example creates a custom policy that rejects properties with null values. The policy is defined in a script named `mypolicy.js`.

```
var policy = {   "policyId" : "notNull",
       "policyExec" : "notNull",
       "policyRequirements" : ["NOT_NULL"]
}

addPolicy(policy);

function notNull(fullObject, value, params, property) {
   if (value == null) {
       return [ {"policyRequirement": "NOT_NULL"}: "NOT_NULL"}];
   }
   return [];
}
```

The policy is referenced in the policy configuration file as follows:

```
{
    "type" : "text/javascript",
    "file" : "bin/defaults/script/policy.js",
    "additionalFiles" : ["script/mypolicy.js"],
    "resources" : [
        {
...
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

You can also configure policies for managed object properties as part of the property definition in the `conf/managed.json` file. For example, the following extract of a managed.json file shows a policy configuration for the `password` property.

```
...
            "properties" : [
                {
                    "name" : "password",
                    "encryption" : {
                        "key" : "openidm-sym-default"
                    },
                    "scope" : "private"
                    "policies" : [
                        {
                            "policyId" : "required"
                        },
                        {
                            "policyId" : "not-empty"
                        },
                        {
                            "policyId" : "at-least-X-capitals",
                            "params" : {
                                "numCaps" : 1
                            }
                        }
                     ]
                  },
...
```

7.3. Disabling Policy Enforcement

Policy enforcement refers to the automatic validation of data in the repository when it is created, updated, or patched. In certain situations you might want to disable policy enforcement temporarily. You might, for example, want to import existing data that does not meet the validation requirements with the intention of cleaning up this data at a later stage.

You can disable policy enforcement by setting `openidm.policy.enforcement.enabled` to `false` in the `conf/boot/boot.properties` file. This setting disables policy enforcement in the back-end only, and has no impact on direct policy validation calls to the Policy Service (which the user interface makes to validate input fields). So, with policy enforcement disabled, data added directly over REST is not subject to validation, but data added with the UI is still subject to validation.

Disabling policy enforcement permanently in a production system is not recommended.

7.4. Managing Policies Over REST

You can manage the policy service over the REST interface, by calling the REST endpoint `http://localhost:8080/openidm/policy`, as shown in the following examples.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

7.4.1. Listing the Defined Policies

The following REST call displays a list of all the defined policies:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/policy"
```

The policy objects are returned in JSON format, with one object for each defined policy ID, for example:

```
{
    "resources": [
        {
            "resource": "managed/user/*",
            "properties": [
                {
                    "policies": [
                        {
                            "policyId": "required",
                            "policyFunction": "function required(fullObject, value, params, propName) {
                                 if (value === undefined) {
                                     return [{"policyRequirement":"REQUIRED"}:"REQUIRED"}];
                                 }
                                   return [];
                         }",
                            "policyRequirements": [
                                "REQUIRED"
                            ]
                        },
                        ...
```

To display the policies that apply to a specific component, include the component name in the URL. For example, the following REST call displays the policies that apply to managed users.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/policy/managed/user/*"
```

```
{
    "resource": "managed/user/*",
    "properties": [
        {
            "policies": [
                {
                    "policyId": "required",
                    "policyFunction": "
                        \nfunction required(fullObject, value, params, propName) {
                            \n    if (value === undefined) {
                            \n        return [{\"policyRequirement\":\"REQUIRED\"}:\"REQUIRED\"}];
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
                            \n    }
                            \n    return [];
                            \n}
                            \n",
                    "policyRequirements": [
                        "REQUIRED"
                    ]
                },
                {
                    "policyId": "not-empty",
                    "policyFunction": "
                    \nfunction notEmpty(fullObject, value, params, property) {
                    \n    if (typeof (value) !== \"string\" || !value.length) {
                    \n        return [{\"policyRequirement\":\"REQUIRED\"}:\"REQUIRED\"}];
                    \n    } else {
                    \n        return [];
                    \n    }
                    \n}
                    \n",
                    "policyRequirements": [
                        "REQUIRED"
                    ]
                },
                {
                    "policyId": "unique",
                    "policyRequirements": [
                        "UNIQUE"
                    ]
                },
...
}
```

7.4.2. Validating Objects and Properties Over REST

Use the `validateObject` action to verify that an object adheres to the requirements of a policy.

The following example verifies that a new managed user object is acceptable in terms of the policy requirements.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 --data '{"familyName":"Jones",
          "givenName":"Bob",
          "_id":"bjones",
          "phoneNumber":"0827878921",
          "passPhrase":null,
          "email":"bjones@example.com",
          "accountStatus":"active",
          "roles":"admin",
          "userName":"bjones@example.com",
          "password":"123"}'
 "http://localhost:8080/openidm/policy/managed/user/bjones?_action=validateObject"

{"result":false,
 "failedPolicyRequirements":[
    {"policyRequirements":[
         {"policyRequirement":"AT_LEAST_X_CAPITAL_LETTERS",
             "params":{"numCaps":1}
         },
         {"policyRequirement":"MIN_LENGTH",
             "params":{"minLength":8}
         }
         ],
      "property":"password"
    }
  ]
}
```

The result (`false`) indicates that the object is not valid. The unfulfilled policy requirements are provided as part of the response - in this case, the user password does not meet the validation requirements.

Use the `validateProperty` action to verify that a specific property adheres to the requirements of a policy.

The following example checks whether Barbara Jensen's new password (`12345`) is acceptable.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 --data '{ "password" : "12345" }'
 "http://localhost:8080/openidm/policy/managed/user/bjensen?_action=validateProperty"

{
    "result": false,
    "failedPolicyRequirements": [
        {
            "policyRequirements": [
                {
                    "policyRequirement": "AT_LEAST_X_CAPITAL_LETTERS",
                    "params": {
                        "numCaps": 1
                    }
                },
                {
                    "policyRequirement": "MIN_LENGTH",
                    "params": {
                        "minLength": 8
                    }
                }
            ],
            "property": "password"
        }
    ]
}
```

The result (`false`) indicates that the password is not valid. The unfulfilled policy requirements are provided as part of the response - in this case, the minimum length and the minimum number of capital letters.

Validating a property that does fulfil the policy requirements returns a `true` result, for example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 --data '{ "password" : "1NewPassword" }'
 "http://localhost:8080/openidm/policy/managed/user/bjensen?_action=validateProperty"

{
    "result": true,
    "failedPolicyRequirements": []
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 8

Configuring Server Logs

This chapter briefly describes server logging. For audit information, see the chapter on *Using Audit Logs*.

You can configure logging by editing the `openidm/conf/logging.properties` file in OpenIDM.

The default configuration writes log messages in simple format to `openidm/logs/openidm*.log` files, rotating files when the size reaches 5 MB, and retaining up to 5 files. Also by default, OpenIDM writes all system and custom log messages to the files.

You can update the configuration to attach loggers to individual packages, setting the log level to one of the following values.

```
SEVERE (highest value)
WARNING
INFO
CONFIG
FINE
FINER
FINEST (lowest value)
```

If you use `logger` functions in your scripts, then you can set the log level for the scripts:

```
org.forgerock.openidm.script.javascript.JavaScript.level=level
```

You can override the log level settings per script by using `org.forgerock.openidm.script.javascript.JavaScript.script-name.level`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 9
Connecting to External Resources

This chapter describes how to connect to external resources such as LDAP, Active Directory, flat files, and others. Configurations shown here are simplified to show essential aspects. Not all resources support all OpenIDM operations, however the resources shown here support most of the CRUD operations, and also reconciliation and LiveSync.

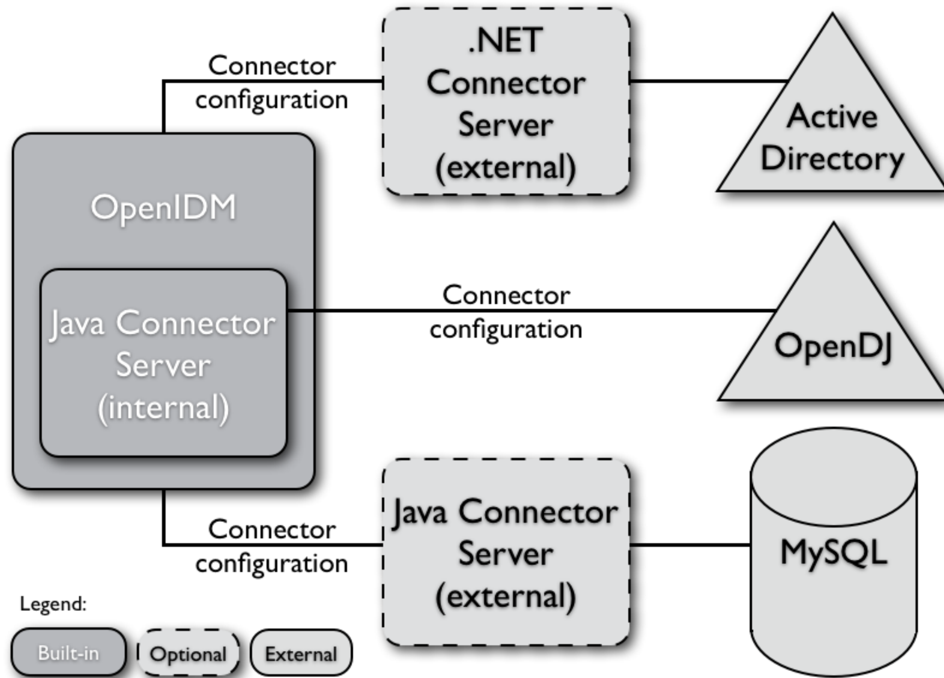
In OpenIDM, *resources* are external systems, databases, directory servers, and other sources of identity data to be managed and audited by the identity management system. OpenIDM connects to resources through the identity connector framework, OpenICF. OpenICF aims to avoid the need to install agents to access resources, instead using the resources' native protocols. For example, OpenICF connects to database resources using the database's Java connection libraries or JDBC driver. It connects to directory servers over JNDI. It connects to UNIX systems by using **ssh**.

Connectors are configured through files named `openidm/conf/provisioner.openicf-name` where *name* corresponds to the name of the connector. *Do not include dash characters (* `-` *) in the connector name.* A number of sample connectors are available in the `openidm/samples/provisioners` directory. To use these connectors, edit the configuration files as required, and copy them to the `openidm/conf` directory.

9.1. About OpenIDM & OpenICF

The following figure shows how OpenIDM can connect to resources through an OpenICF server. In most cases, the OpenICF server runs as part of OpenIDM.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.



OpenICF provides a common service provider interface to allow identity services access to the resources that contain user information. OpenICF uses a connection server that can run as a local connector server inside OpenIDM, or as a remote connector server that is a standalone process.

A remote connector server is required when access libraries that cannot be included as part of the OpenIDM process are needed. If a resource, such as Microsoft ADSI, does not provide a connection library that can be included inside the Java Virtual Machine, then OpenICF can use the native .dll with a remote .NET connector server. (OpenICF connects to ADSI through a remote connector server implemented as a .NET service.)

> **Tip**
>
> Not only .NET connector servers but also Java connector servers can be run as standalone, remote services. Run them as remote services for scalability, or to have the service run in the cloud.
>
> By default, and for convenience, OpenIDM includes a Java connector server that runs as a `"#LOCAL"` service.

9.2. Accessing Remote Connectors

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

When you configure remote connectors, you must use the connector info provider service to connect through remote connector servers. The configuration is stored in the configuration file, `openidm/conf/provisioner.openicf.connectorinfoprovider.json`. A sample can be found under `openidm/samples/provisioners/`. To use the file, edit it as required, and then copy it to the `openidm/conf` directory.

The connector info provider service takes the following configuration:

```
{
  "connectorsLocation"     : string,
  "remoteConnectorServers" : [remoteConnectorServer objects]
}
```

Connector Info Provider Properties

connectorsLocation

string, optional

Specifies the directory where the OpenICF connectors are located. The default location is `openidm/connectors`.

remoteConnectorServers

array of RemoteConnectorServer objects, optional

A list of remote connector servers managed by this service.

Remote Connector Server Properties

The following example shows a `remoteConnectorServer` object configuration.

```
{
  "name"                : "testServer",
  "host"                : "127.0.0.1",
  "port"                : 8759,
  "heartbeatInterval"   : 60,
  "useSSL"              : false,
  "timeout"             : 0,
  "key"                 : "changeit",
  "trustManagers"       :
    [
       "X509TrustManager",
       "BlindTrustManager"
    ]
}
```

OpenIDM supports the following remote connector server object properties.

name

string, required

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The name of the remote connector server object. Used to identify the remote connector server in connector reference objects.

host

string, required

Remote host to connect to.

port

string, optional

Remote port to connect to. Default value: 8759

heartbeatInterval

integer, optional

Specifies the interval, in seconds, at which heartbeat packets are transmitted. If the connector server is unreachable, based on this heartbeat interval, all services that use the connector server are made unavailable until the connector server can be reached again. Default value: 60

useSSL

boolean, optional

Specifies whether or not to use SSL to connect. Default value: `false`

timeout

integer, optional

Specifies the timeout (in milliseconds) to use for the connection. Default value: 0

key

string, required

The secret key to use to authenticate to the remote connector server.

trustManagers

not specified

Not implemented yet. The service uses the default JVM `TrustManager`.

9.3. Configuring Connectors

Connectors are configured through the OpenICF provisioner service. Each connector configuration is stored in a file in the `openidm/conf/` folder or under the same URL respectively. Configuration files are

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

named `openidm/conf/provisioner.openicf-name` where *name* corresponds to the name of the connector. *Do not include dash characters (`-`) in the connector name.* A number of sample connectors are available in the `openidm/samples/provisioners` directory. To use these connectors, edit the configuration files as required, and copy them to the `openidm/conf` directory.

The following example shows an OpenICF provisioner service configuration for an XML file resource.

```
{
 "name"                       : "xml",
 "connectorRef"               : connector-ref-object,
 "poolConfigOption"           : pool-config-option-object,
 "operationTimeout"           : operation-timeout-object,
 "configurationProperties"    : configuration-properties-object,
 "objectTypes"                : object-types-object,
 "operationOptions"           : operation-options-object
}
```

Connector Reference

The following example shows a connector reference object.

```
{
  "bundleName"       : "org.forgerock.openicf.connectors.file.xml",
  "bundleVersion"    : "1.1.1.0",
  "connectorName"    : "com.forgerock.openicf.xml.XMLConnector",
  "connectorHostRef" : "host"
}
```

bundleName

string, required

The *ConnectorBundle-Name* of the OpenICF connector.

bundleVersion

string, required

The *ConnectorBundle-Version* of the OpenICF connector.

connectorName

string, required

The Connector implementation class name.

connectorHostRef

string, optional

The name of the RemoteConnectorServer object.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

- If the connector server is local and the connector .jar is installed in `openidm/bundle/` (currently not recommended), then the value must be `"osgi:service/org.forgerock.openicf.framework.api.osgi.ConnectorManager"`.
- If the connector server is local and the connector .jar is installed in `openidm/connectors/`, then the value must be `"#LOCAL"`. This is currently the default location.

Pool Configuration Option

The following example shows a pool configuration option object for the connection pool between OpenIDM and the OpenICF connector server.

```
{
  "maxObjects"                 : 10,
  "maxIdle"                    : 10,
  "maxWait"                    : 150000,
  "minEvictableIdleTimeMillis" : 120000,
  "minIdle"                    : 1
}
```

maxObjects

Maximum number of idle and active objects.

maxIdle

Maximum number of idle objects

maxWait

The maximum time in milliseconds which the pool waits for an object before timing out. Zero means never time out.

minEvictableIdleTimeMillis

Maximum time in milliseconds an object can be idle before it is removed. Zero means never time out.

minIdle

The minimum number of idle objects.

Operation Timeout

This configuration sets the timeout per operation type.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
  "CREATE"              : -1,
  "TEST"                : -1,
  "AUTHENTICATE"        : -1,
  "SEARCH"              : -1,
  "VALIDATE"            : -1,
  "GET"                 : -1,
  "UPDATE"              : -1,
  "DELETE"              : -1,
  "SCRIPT_ON_CONNECTOR" : -1,
  "SCRIPT_ON_RESOURCE"  : -1,
  "SYNC"                : -1,
  "SCHEMA"              : -1
}
```

operation-name

Timeout in milliseconds

A value of `-1` disables the timeout.

Configuration Properties

This object contains the configuration for the connection between the connection server and the resource, and is therefore resource specific.

The following example shows a configuration properties object for the default XML sample resource connector.

```
{
  "xsdIcfFilePath": "samples/sample1/data/resource-schema-1.xsd",
  "xsdFilePath": "samples/sample1/data/resource-schema-extension.xsd",
  "xmlFilePath": "samples/sample1/data/xmlConnectorData.xml"
 }
```

property

Individual properties depend on the type of connector.

Object Types

This configuration object specifies the supported object types. The property name defines the `objectType` used in the URI: `system/$systemName/$objectType`

The configuration is based on JSON Schema with extensions described below.

Attribute names which start and/or end with `__` are resource type specific attributes used by OpenICF for particular purposes, such as `__NAME__` as the naming attribute for objects on a resource.

```
{
  "account" :
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
  {
    "$schema" : "http://json-schema.org/draft-03/schema",
    "id" : "__ACCOUNT__",
    "type" : "object",
    "nativeType" : "__ACCOUNT__",
    "properties" :
    {
      "name" :
      {
        "type" : "string",
        "nativeName" : "__NAME__",
        "nativeType" : "JAVA_TYPE_PRIMITIVE_LONG",
        "flags" :
        [
          "NOT_CREATABLE",
          "NOT_UPDATEABLE",
          "NOT_READABLE",
          "NOT_RETURNED_BY_DEFAULT"
        ]
      },
      "groups" :
      {
        "type" : "array",
        "items" :
        {
          "type" : "string",
          "nativeType" : "string"
        },
        "nativeName" : "__GROUPS__",
        "nativeType" : "string",
        "flags" :
        [
          "NOT_RETURNED_BY_DEFAULT"
        ]
      },
      "givenName" : {
         "type" : "string",
         "nativeName" : "givenName",
         "nativeType" : "string"
         },
    }
  }
}
```

Object Level Extensions

nativeType

string, optional

The native OpenICF object type.

Property Level Extensions

nativeType

string, optional

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The native OpenICF attribute type.

nativeName

string, optional

The native OpenICF attribute name.

flags

string, optional

The native OpenICF attribute flags. The *required* and *multivalued* flags are defined by the JSON schema.

```
required = "required" : true
```

```
multivalued = "type" : "array"
```

Note

Avoid using the dash character (`-`) in property names, like `last-name`, as dashes in names make JavaScript syntax more complex. If you cannot avoid the dash, then write `source['last-name']` instead of `source.last-name` in the java script scripts.

Operation Options

Operation options define how to act on specified operations. You can for example deny operations on specific resources to avoid OpenIDM accidentally updating a read-only resource during a synchronization operation.

```
{
  "SYNC" :
  {
    "denied" : true,
    "onDeny" : "DO_NOTHING",
    "objectFeatures" :
    {
      "__ACCOUNT__" :
      {
        "denied" : true,
        "onDeny" : "THROW_EXCEPTION",
        "operationOptionInfo" :
        {
          "$schema" : "http://json-schema.org/draft-03/schema",
          "id" : "FIX_ME",
          "type" : "object",
          "properties" :
          {
            "_OperationOption-float" :
            {
               "type" : "number",
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
                "nativeType" : "JAVA_TYPE_PRIMITIVE_FLOAT"
             }
           }
         }
       },
       "__GROUP__" :
       {
         "denied" : false,
         "onDeny" : "DO_NOTHING"
       }
     }
   }
}
```

The list of operations is as follows.

- `AUTHENTICATE`: AuthenticationApiOp
- `CREATE`: CreateApiOp
- `DELETE`: DeleteApiOp
- `GET`: GetApiOp
- `RESOLVEUSERNAME`: ResolveUsernameApiOp
- `SCHEMA`: SchemaApiOp
- `SCRIPT_ON_CONNECTOR`: ScriptOnConnectorApiOp
- `SCRIPT_ON_RESOURCE`: ScriptOnResourceApiOp
- `SEARCH`: SearchApiOp
- `SYNC`: SyncApiOp
- `TEST`: TestApiOp
- `UPDATE`: UpdateApiOp
- `VALIDATE`: ValidateApiOp

denied

boolean, optional

This property prevents operation execution if the value is `true`.

onDeny

string, optional

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

If `denied` is `true`, then the service uses this value. Default value: `DO_NOTHING`.

- `DO_NOTHING`: On operation the service does nothing.
- `THROW_EXCEPTION`: On operation the service throws a `ForbiddenException` exception.

9.4. Connector Configuration Examples

This section explains provisioner configurations for common connectors. Also see Section 9.5, "Creating Default Connector Configurations" for instructions on interactively building connector configurations.

9.4.1. XML File Connector

The following example shows an excerpt of the provisioner configuration for an XML file connector.

```
{
    "connectorRef": {
        "connectorHostRef": "#LOCAL",
        "bundleName":
            "org.forgerock.openicf.connectors.file.file.openicf-xml-connector",
        "bundleVersion": "1.1.1.0",
        "connectorName": "com.forgerock.openicf.xml.XMLConnector"
    }
}
```

The connectorHostRef is optional if the connector server is local.

The configuration properties for the XML file connector set the relative path to the file containing the identity data, and also the paths to the XML schemas required.

```
{
    "configurationProperties": {
        "xsdIcfFilePath": "samples/sample1/data/resource-schema-1.xsd",
        "xsdFilePath": "samples/sample1/data/resource-schema-extension.xsd",
        "xmlFilePath": "samples/sample1/data/xmlConnectorData.xml"
    }
}
```

xmlFilePath

References the XML file containing account entries

xsdIcfFilePath

References the XSD file defining schema common to all XML file resources. Do not change the schema defined in this file.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

xsdFilePath

References custom schema defining attributes specific to your project

9.4.2. Generic LDAP Connector

The following excerpt shows the `connectorRef` configuration property for connection to an LDAP server. When using the connect .jar provided in `openidm/connectors`, and when using a local connector server, the `connectorHostRef` property is optional.

```
{
    "connectorRef": {
        "connectorHostRef": "#LOCAL",
        "connectorName": "org.identityconnectors.ldap.LdapConnector",
        "bundleName":
            "org.forgerock.openicf.connectors.ldap-connector",
        "bundleVersion": "1.1.1.0"
    }
}
```

The following excerpt shows settings for many connector configuration properties.

```
{
    "accountSynchronizationFilter": null,
    "passwordAttributeToSynchronize": null,
    "synchronizePasswords": false,
    "removeLogEntryObjectClassFromFilter": true,
    "modifiersNamesToFilterOut": [],
    "passwordDecryptionKey": null,
    "credentials": "Passw0rd",
    "changeLogBlockSize": 100,
    "baseContextsToSynchronize": [
        "ou=People,dc=example,dc=com"
    ],
    "attributesToSynchronize": [
        "uid",
        "sn",
        "cn",
        "givenName",
        "mail",
        "description"
    ],
    "changeNumberAttribute": "changeNumber",
    "passwordDecryptionInitializationVector": null,
    "filterWithOrInsteadOfAnd": false,
    "objectClassesToSynchronize": [
        "inetOrgPerson"
    ],
    "port": 1389,
    "vlvSortAttribute": "uid",
    "passwordAttribute": "userPassword",
    "useBlocks": true,
    "maintainPosixGroupMembership": false,
    "failover": [],
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
    "ssl": false,
    "principal": "cn=Directory Manager",
    "baseContexts": [
        "dc=example,dc=com"
    ],
    "readSchema": true,
    "accountObjectClasses": [
        "top",
        "person",
        "organizationalPerson",
        "inetOrgPerson"
    ],
    "accountUserNameAttributes": [
        "uid",
        "cn"
    ],
    "host": "localhost",
    "groupMemberAttribute": "uniqueMember",
    "accountSearchFilter": null,
    "passwordHashAlgorithm": null,
    "usePagedResultControl": false,
    "blockSize": 100,
    "uidAttribute": "entryUUID",
    "maintainLdapGroupMembership": false,
    "respectResourcePasswordPolicyChangeAfterReset": false
}
```

accountSynchronizationFilter

Used during synchronization actions to filter out LDAP accounts

accountObjectClasses

The object classes used when creating new LDAP user objects. When specifying more than one object class, add each object class as its own property. For object classes that inherit from parents other than `top`, such as `inetOrgPerson`, specify all object classes in the class hierarchy.

accountSearchFilter

Search filter that accounts must match

accountUserNameAttributes

Attributes holding the account's user name. Used during authentication to find the LDAP entry matching the user name.

attributesToSynchronize

List of attributes used during object synchronization. OpenIDM ignores change log updates that do not include any of the specified attributes. If empty, OpenIDM considers all changes.

baseContexts

Base DNs for operations on the LDAP server

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

baseContextsToSynchronize

Base DNs for entries taken into account during synchronization

blockSize

Block size for simple paged results and VLV index searches, reflecting the maximum number of accounts retrieved at any one time

changeLogBlockSize

Block size used when fetching change log entries

changeNumberAttribute

Change log attribute containing the last change number

credentials

Password to connect to the LDAP server

failover

LDAP URLs specifying alternative LDAP servers to connect to if OpenIDM cannot connect to the primary LDAP server specified in the `host` and `port` properties

filterWithOrInsteadOfAnd

In most cases, the filter to fetch change log entries is AND-based. If this property is set, the filter ORs the required change numbers instead.

groupMemberAttribute

LDAP attribute holding members for non-POSIX static groups

host

Primary LDAP server host name

maintainLdapGroupMembership

If `true`, OpenIDM modifies group membership when entries are renamed or deleted.

maintainPosixGroupMembership

If `true`, OpenIDM modifies POSIX group membership when entries are renamed or deleted.

modifiersNamesToFilterOut

Use to avoid loops caused by OpenIDM's own changes

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

objectClassesToSynchronize

OpenIDM synchronizes only entries having these object classes.

passwordAttribute

Attribute to which OpenIDM writes the predefined PASSWORD attribute

passwordAttributeToSynchronize

OpenIDM synchronizes password values on this attribute.

passwordDecryptionInitializationVector

Initialization vector used to decrypt passwords when performing password synchronization

passwordDecryptionKey

Key used to decrypt passwords when performing password synchronization

passwordHashAlgorithm

Hash password values with the specified algorithm if the LDAP server stores them in clear text

port

Primary LDAP server port number

principal

Bind DN used to connect to the LDAP server

readSchema

If `true`, read LDAP schema from the LDAP server.

removeLogEntryObjectClassFromFilter

If `true`, the filter to fetch change log entries does not contain the `changeLogEntry` object class, and OpenIDM expects no entries with other object types in the change log. Default: `true`

respectResourcePasswordPolicyChangeAfterReset

If `true`, bind with the Password Expired and Password Policy controls, and throw `PasswordExpiredException` and other exceptions appropriately.

ssl

If `true`, the specified port listens for LDAPS connections.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

synchronizePasswords

If `true`, synchronize passwords.

uidAttribute

OpenIDM maps `uid` to the specified attribute.

useBlocks

If `true`, use block-based LDAP controls like simple paged results and virtual list view.

usePagedResultControl

If `true`, use simple paged results rather than virtual list view when both are available.

vlvSortAttribute

Attribute used as the sort key for virtual list view

9.4.3. Active Directory Connector

In contrast to most other connectors, the Active Directory connector is written not in Java, but instead in .NET. OpenICF should connect to Active Directory over ADSI, the native connection protocol for Active Directory. The connector therefore requires a connector server that has access to the ADSI .dll files.

9.4.3.1. Installing and Configuring a .NET Connector

The use of .NET connector servers is useful when an application is written in Java, but a connector bundle is written using C#. Since a Java application (e.g. J2EE application) cannot load C# classes, it is necessary to deploy the C# bundles under a .NET connector server. The Java application can communicate with the C# connector server over the network, and the C# connector server serves as a proxy to provide to any authenticated application access to the C# bundles deployed within the C# connector server.

Make sure the following requirements are met prior to installing .NET.

Procedure 9.1. Installing the .NET Connector Server

1. Download the OPENICF .NET Connector Server from the OpenIDM download page under the ForgeRock Open Stack download page.

2. Execute ServiceInstall-1.1.1.0-dotnet.msi.

3. Complete the wizard.

When the wizard has run, the Connector Server is installed as a Windows Service.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Procedure 9.2. Running the .NET Connector Server

The .NET Connector Server can be started one of two ways.

1. In the Microsoft Service Console, go to `Start`, type `Services`, `Services`.

   or

2. In the command prompt, go to `Start`, type `cmd`, then `cmd` again.

3. Change the directory to the location where the Connector Server was installed. The default location is `Program Files/Identity Connectors/Connector Server`.

4. Enter `cd Program Files (x86)/Identity Connectors/Connector Server`.

5. Start the server with the following command:

   ```
   ConnectorServer.exe/run
   ```

   .

Procedure 9.3. Configuring the .NET Connector Server

After starting the Microsoft Services Console, follow these steps to configure the .NET Connector Server.

1. Check to see if the Connector Server is currently running. If so, stop it. All configuration changes require that the Connector Server be stopped and restarted after the changes are saved.

2. At the command prompt (click `Start`, `Run`, then type `cmd`, set the key for the Connector Server by changing to the directory where the Connector Server was installed and executing the following command, using a string value for `newkey`:

   ```
   ConnectorServer.exe /setkey <newkey>
   ```

   This key is used by clients connecting to the Connector Server.

3. Review the `ConnectorServer.exe.config` file to verify additional configuration. This includes the port, address, and SSL settings in the `AppSettings`.

   ```
   <add key="connectorserver.port" value="8759" />
   <add key="connectorserver.usessl" value="false" />

   <add key="connectorserver.certificatestorename" value="ConnectorServerSSLCertificate" />
   <add key="connectorserver.ifaddress" value="0.0.0.0" />
   ```

   The port can be set by changing the value of `connectorserver.port`. The listening socket can be bound to a particular address, or can be left as 0.0.0.0.

   To configure the server to use SSL, set the value of `connectorserver.usessl` to `true` and set the value of `connectorserver.certifacatestorename` to the certificate store name.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

4. Trace settings are also in the configuration file.

```
<system.diagnostics>
<trace autoflush="true" indentsize="4">
<listeners>
<remove name="Default" />
<add name="myListener" type="System.Diagnostics.TextWriterTraceListener"
initializeData="c:\connectorserver2.log" traceOutputOptions="DateTime">
<filter type="System.Diagnostics.EventTypeFilter" initializeData="Information" />
</add>
</listeners>
</trace>
</system.diagnostics>
```

The Connector Server uses the standard .NET trace mechanism. For more information about the tracing options, see Microsoft's .NET documentation for `System.Diagnostics`.

> **Note**
>
> The default settings are a good starting point, but for less tracing, you can change the EventTypeFilter's initializeData to "Warning" or "Error". For very verbose logging you can set the value to "Verbose" or "All". The amount of logging performed has a direct effect on the performance of the Connector Servers, so be careful of the setting.

5. Download the AD Connector from the OpenIDM download page under the ForgeRock Open Stack download page.
6. Unzip the directory in the Connector Server folder.
7. Start the Connector Server service.

Procedure 9.4. Configuring the .NET Connector Server with OpenIDM

When you configure remote connectors, you must use the connector info provider service to connect through remote connector servers. The configuration is stored in the configuration file, `openidm/conf/provisioner.openicf.connectorinfoprovider.json`. A sample can be found under `openidm/samples/provisioners/`.

1. Make sure that OpenIDM is running and copy the `provisioner.openicf.connectorinfoprovider.json` to `/path/to/openidm/conf` and edit it as needed.

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

2. Create the connector file `provisioner.openicf-ad.json` in `conf/ directory`. The following is an example of what the `name`, `bundleVersion`, and a few other configuration properties will look like.

```

      {
"name" : "ad",
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
"connectorRef" : {
"connectorHostRef" : "dotnet",
"connectorName" : "Org.IdentityConnectors.ActiveDirectory.ActiveDirectoryConnector",
"bundleName" : "ActiveDirectory.Connector",
"bundleVersion" : "1.0.0.0"
},
"poolConfigOption" : {
"maxObjects" : 10,
"maxIdle" : 10,
"maxWait" : 150000,
"minEvictableIdleTimeMillis" : 120000,
"minIdle" : 1
},
"operationTimeout" : {
"SYNC" : -1,
"TEST" : -1,
"SEARCH" : -1,
"RESOLVEUSERNAME" : -1,
"SCRIPT_ON_CONNECTOR" : -1,
"VALIDATE" : -1,
"DELETE" : -1,
"UPDATE" : -1,
"AUTHENTICATE" : -1,
"CREATE" : -1,
"SCRIPT_ON_RESOURCE" : -1,
"GET" : -1,
"SCHEMA" : -1
},
"configurationProperties" : {
"DirectoryAdminName" : "EXAMPLE\\Administrator",
"DirectoryAdminPassword" : {
"$crypto" : {
"value" : {
"iv" : "QJctjWJi9w2uPLsO2Pucfw==",
"data" : "Akqzk1PW0m9QP5cfOMIuYw==",
"cipher" : "AES/CBC/PKCS5Padding",
"key" : "openidm-sym-default"
},
"type" : "x-simple-encryption"
}
},
"ObjectClass" : "User",
"Container" : "dc=example,dc=com",
"CreateHomeDirectory" : true,
"LDAPHostName" : "10.0.0.2",
"SearchChildDomains" : false,
"DomainName" : "example",
"SyncGlobalCatalogServer" : null,
"SyncDomainController" : null,
"SearchContext" : ""
}
```

3. Edit the `configurationProperties` according to your setup and make sure that the `bundleVersion` is the same version as `ActioveDirectory.Connector.dll` in the Windows Connector Server folder. (Right click on the `dll`, `properties`, `tab details`, and `Product version`.)

4. Make sure the connector was installed properly using the following command:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
scr list
```

This should return all of the installed modules, including the following:

```
[  24] [active       ] org.forgerock.openidm.provisioner.openicf
```

> **Note**
>
> The number may differ. Make sure to note the number returned.

Review the content of the connector using the following command, using the number returned from the previous step:

```
scr info <your number>
```

5. Create the `sync.json` file where you define mappings of various attributes and behaviors during reconciliation. The following is a simple example of a `sync.json`.

```
{
"mappings" : [
{
"name" : "systemADAccounts_managedUser",
"source" : "system/ad/account",
"target" : "managed/user",
"properties" : [
 {
   "source" : "sAMAccountName",
    "target" : "userName"
 },
 {
    "source" : "sn",
    "target" : "lastname"
 },
 {
    "source" : "givenName",
    "target" : "firstname"
    }
   ]
  }
 ]
}
```

6. Run the reconciliation with the following command.

```
$ curl --header "X-OpenIDM-Username: openidm-admin" --header "X-OpenIDM-Password:
openidm-admin" --request POST  "http://localhost:8080/openidm/recon?
_action=recon&mapping=systemADAccounts_managedUser"
```

This will return a reconciliation id similar to the following:

```
{"_id":"0629d920-e29f-4650-889f-4423632481ad"}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

7. Check the internal repository (OrientDB or MySQL) to make sure that the users were reconciled. For information about connecting to OrientDB, see *Before You Begin* in the *Installation Guide* in the *Installation Guide*. For information about using MySQL as a repository, see *Installing a Repository For Production* in the *Installation Guide* in the *Installation Guide*.

9.4.3.2. Installing a Standalone Java Connector Server

It may be necessary to set up a remote Java Connector Server (JSC). This section provides directions for setting up the standalone connecter on Unix/Linux and Windows.

Procedure 9.5. Installing a Standalone Connector Server for Unix/Linux

1. Download the OPENICF JAVA Connector Server from the OpenIDM download page under the ForgeRock Open Stack download page.
2. Run the terminal and unpack it. The following command will unzip the file in the current folder, so make sure to move to the appropriate location prior to running the command.

```
unzip openicf-1.1.1.0-java.zip
```

3. Change the directory to OpenICF using the following command:

```
$ cd path/to/openicf
```

4. If needed, secure the communication between OpenIDM and JCS. The JCS uses a property called `secret key` to authenticate the connection. The default secret key value is `changeit`. To change the value of the secret key enter the following, replacing `newkey` with the your own string value:

```
java - cp "./lib/framework/*" org.identityconnectors.framework.server.Main
       -setKey -key <newkey> -properties ./conf/ConnectorServer.properties
```

5. Review the `ConnectorServer.properties` in the `/conf` directory, and make changes as necessary. The file contains setting information, including things like ports, the allowance of one/all IP addresses, and SSL. The file provides the required information to update these settings.
6. Run the JCS.

```
java -cp "./lib/framework/*"
org.identityconnectors.framework.server.Main -run -properties
./conf/ConnectorServer.properties
```

7. If necessary, you can stop the JCS by pressing `^C`.

Procedure 9.6. Installing a Standalone Connector Server for Windows

1. Download the OPENICF JAVA Connector Server from the OpenIDM download page under the ForgeRock Open Stack download page.
2. Unpack the zip file in the desired location, for example `C:\openicf`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

3. Run the command line (`Start`, type `cmd`, and `cmd`) and change the working directory to `openicf\bin cd c:\openicf\bin`
4. Change the directory to OpenICF using the following command:

```
$ cd path/to/openicf
```

5. If needed, secure the communication between OpenIDM and JCS. The JCS uses a property called `secret key` to authenticate the connection. The default secret key value is `changeit`.

   To change the value of the secret key enter the following, replacing `newkey` with the your own string value:

```
/ConnectorServer.bat /setkey <newkey>
```

6. Review the `ConnectorServer.properties` in the `/conf` directory, and make changes as necessary. The file contains setting information, including things like ports, the allowance of one/all IP addresses, and SSL. The file provides the required information to update these settings.
7. If you would like the JCS to run as a Windows service, enter the following command.

```
./ConnecorServer.bat /install (for uninstalling use /uninstall )
```

> **Note**
>
> If you install JCS as a Windows service you can start/stop it by Microsoft's Service Console (`Start`, type `Service`, and `Service`). The JCS service is called: `OpenICFConnectorServerJava`.

   Or

   If you would like to run the JCS from command line, enter the following command:

```
.\ConnectorServer.bat /run
```

8. If necessary, stop the JCS by pressing `^C`.

You can view the log files in the `openicf/logs` directory.

9.4.3.2.1. MySQL Database Example to Reconcile JCS Users

This sample demonstrates using reconciliation of users stored in a MySQL database on a remote machine. The JCS runs on the same machine as the MySQL database and mediates the connection between OpenIDM and MySQL database.

Procedure 9.7. Configuring JCS

1. Download MySQL JDBC Driver.
2. Unpack the `MySQL JDBC Driver` and copy the `mysql-connector-java-5.1.22-bin.jar` to the `openicf/lib` directory.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

3. Go to the `/tools` directory. The groovy scripts in the folder run on the JCS side. Copy them from `path/to/openidm/sample/sample3/tools` folder to `openicf/`.

Procedure 9.8. Configuring OpenIDM for the MySQL Database Example

1. Start OpenIDM. You can ignore errors like `cannot connect to database` and `cannot find jdbc driver`. These errors will be fixed once OpenIDM is configured and restarted.

2. Go to the `provisioner.openicf.connectorinfoprovider.json` to see information about your remote connector servers. Copy this file from `openidm/samples/provisioners` to `openidm/conf`.

   For Unix/Linux, enter the following in Terminal.

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

   For Windows, enter the following on the command line.

```
c:\> cd path/to/openidm
.\> copy .\samples\provisioners\provisioner.openicf.connectorinfoprovider.json .\conf
```

3. Edit the `provisioner.openicf.connectorinfoprovider.json` to meet your needs. The following is an example.

```
{
     "connectorsLocation" : "connectors",
     "remoteConnectorServers" : [
         {
             "name" : "mysql",
             "host" : "10.0.0.2",
             "port" : 8759,
             "useSSL" : false,
             "timeout" : 0,
             "key" : "password"
         }
     ],
 }
```

4. Copy all of the files from `openidm/samples/sample3/conf` to `openidm/conf`.

   For Unix/Linux, enter the following in Terminal.

```
$ cp -r ./samples/sample3/conf ./conf
```

   For Window, enter the following on the command line.

```
.\> copy .\samples\sample3\conf\ .\conf\
```

5. Edit the `provisioner.openicf-scriptedsql.json` to read like the following.

```
{
     "name" : "hrdb",
     "connectorRef" : {
         "connectorHostRef" : "mysql",
         "bundleName" : "org.forgerock.openicf.connectors.db.openicf-scriptedsql
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
        -connector",
        "bundleVersion" : "1.1.1.0",
        "connectorName" : "org.forgerock.openicf.scriptedsql.ScriptedSQLConnector"
    },
    "producerBufferSize" : 100,
    "connectorPoolingSupported" : true,
    "poolConfigOption" : {
        "maxObjects" : 10,
        "maxIdle" : 10,
        "maxWait" : 150000,
        "minEvictableIdleTimeMillis" : 120000,
        "minIdle" : 1
    },
    "operationTimeout" : {
        "CREATE" : -1,
        "TEST" : -1,
        "AUTHENTICATE" : -1,
        "SEARCH" : -1,
        "VALIDATE" : -1,
        "GET" : -1,
        "UPDATE" : -1,
        "DELETE" : -1,
        "SCRIPT_ON_CONNECTOR" : -1,
        "SCRIPT_ON_RESOURCE" : -1,
        "SYNC" : -1,
        "SCHEMA" : -1
    },
    "configurationProperties" : {
        "host" : "10.0.0.2",
        "port" : "3306",
        "user" : "root",
        "password" : {
            "$crypto" : {
                "value" : {
                    "iv" : "dsrEhCU45UakY6Uh9Jxfww==",
                    "data" : "X1+77+0I7Yog/6ZirsFSyg==",
                    "cipher" : "AES/CBC/PKCS5Padding",
                    "key" : "openidm-sym-default"
                },
                "type" : "x-simple-encryption"
            }
        },
        "database" : "HRDB",
        "autoCommit" : true,
        "reloadScriptOnExecution" : false,
        "keyColumn" : "uid",
        "jdbcDriver" : "com.mysql.jdbc.Driver",
        "jdbcConnectionUrl" : "jdbc:mysql://10.0.0.2:3306/HRDB",
        "jdbcUrlTemplate" : "jdbc:mysql://%h:%p/%d",
        "createScriptFileName" : "/home/tester/openicf/tools/CreateScript.groovy",
        "testScriptFileName" : "/home/tester/openicf/tools/TestScript.groovy",
        "searchScriptFileName" : "/home/tester/openicf/tools/SearchScript.groovy",
        "deleteScriptFileName" : "/home/tester/openicf/tools/DeleteScript.groovy",
        "updateScriptFileName" : "/home/tester/openicf/tools/UpdateScript.groovy",
        "syncScriptFileName" : "/home/tester/openicf/tools/SyncScript.groovy"
    },
    "objectTypes" : {
        "group" : {
            "$schema" : "http://json-schema.org/draft-03/schema",
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
            "id" : "__GROUP__",
            "type" : "object",
            "nativeType" : "__GROUP__",
            "properties" : {
                "name" : {
                    "type" : "string",
                    "required" : true,
                    "nativeName" : "__NAME__",
                    "nativeType" : "string"
                },
                "gid" : {
                    "type" : "string",
                    "required" : true,
                    "nativeName" : "gid",
                    "nativeType" : "string"
                },
                "description" : {
                    "type" : "string",
                    "required" : false,
                    "nativeName" : "description",
                    "nativeType" : "string"
                }
            }
        },
        "organization" : {
            "$schema" : "http://json-schema.org/draft-03/schema",
            "id" : "organization",
            "type" : "object",
            "nativeType" : "organization",
            "properties" : {
                "name" : {
                    "type" : "string",
                    "required" : true,
                    "nativeName" : "__NAME__",
                    "nativeType" : "string"
                },
                "description" : {
                    "type" : "string",
                    "required" : false,
                    "nativeName" : "description",
                    "nativeType" : "string"
                }
            }
        },
        "account" : {
            "$schema" : "http://json-schema.org/draft-03/schema",
            "id" : "__ACCOUNT__",
            "type" : "object",
            "nativeType" : "__ACCOUNT__",
            "properties" : {
                "firstName" : {
                    "type" : "string",
                    "nativeName" : "firstname",
                    "nativeType" : "string",
                    "required" : true
                },
                "email" : {
                    "type" : "array",
                    "items" : {
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
                        "type" : "string",
                        "nativeType" : "string"
                    },
                    "nativeName" : "email",
                    "nativeType" : "string"
                },
                "__PASSWORD__" : {
                    "type" : "string",
                    "nativeName" : "password",
                    "nativeType" : "JAVA_TYPE_GUARDEDSTRING",
                    "flags" : [
                        "NOT_READABLE",
                        "NOT_RETURNED_BY_DEFAULT"
                    ]
                },
                "uid" : {
                    "type" : "string",
                    "nativeName" : "__NAME__",
                    "required" : true,
                    "nativeType" : "string"
                },
                "fullName" : {
                    "type" : "string",
                    "nativeName" : "fullname",
                    "nativeType" : "string"
                },
                "lastName" : {
                    "type" : "string",
                    "required" : true,
                    "nativeName" : "lastname",
                    "nativeType" : "string"
                },
                "organization" : {
                    "type" : "string",
                    "required" : true,
                    "nativeName" : "organization",
                    "nativeType" : "string"
                }
            }
        }
    },
    "operationOptions" : { }
}
```

6. Verify that the following settings are correct.

   - The value of `connectorHostRef : mysql` points to the property `name` of `provisioner.openicf.connectorinfoprovider.json`. This indicates which connectorinfoprovider to use.
   - The `bundleVersion : 1.1.1.0` must be exactly the same as `openicf-scriptedsql-connector-1.1.1.0.jar`. on JCS `/bundles`. Unpack the `.jar` file, open `META-INF/MANIFEST.MF`, and search for the `Bundle-Version` property.
   - The path to groovy scripts should be `createScriptFileName : /home/tester/openicf/tools/CreateScript.groovy ....`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

For Windows, the path will follow Unix notation. For example the path could be `Program Files (x86)/openicf/tools/CreateScript.groovy`, which in Windows notation would be `C:\Program Files (x86)\openicf\tools\CreateScript.groovy`.

- All instances of the connection setting must be properly set, for example, `jdbcConnectionUrl : jdbc:mysql://10.0.0.2:3306/HRDB`.

7. Restart OpenIDM to verify that all of the configuration changes have occurred. There should be no error message when OpenIDM is restarted. To check run the following.

```
src list
```

This returns a list of installed modules, including the following:

```
[  17] [active       ] org.forgerock.openidm.provisioner.openicf
```

Note

The number may differ. Make sure to note the number returned.

When you have installed more connectors, there will be more OpenICF modules. If the state of the module is active, the module is installed properly. If the state is unsatisfied, then you have not configured it correctly and you must check your configuration. You can also check the content of installed module (which could be handy if you have unsatisfied state and you want to see if the content is the same as in `*.json` – to verify that the configuration you just set was picked up). To list the content of the module use the following with the number returned from the previous step:

```
scr info 17 <your number>
```

Note

You can also check the `provisioner.openicf.connectorinfoprovider`

8. Run reconciliation with the following command:

```
$ curl --header "X-OpenIDM-Username: openidm-admin"
--header "X-OpenIDM-Password: openidm-admin"  --request POST
"http://localhost:8080/openidm/recon?_action=recon&mapping=systemHrdb_managedUser"
```

This will return a reconciliation id similar to the following:

```
{"_id":"a5346543-db9a-4f8b-ba25-af2a1b576a54"}
```

9. Check the internal repository (OrientDB or MySQL) to make sure that the users were reconciled. For information about connecting to OrientDB, see *Before You Begin* in the *Installation Guide* in the *Installation Guide*. For information about using MySQL as a repository, see *Installing a Repository For Production* in the *Installation Guide* in the *Installation Guide*.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

9.4.3.3. XML Example to Reconcile JCS Users

This sample demonstrates using reconciliation of users created in an XML folder on a remote machine. The JCS provides a way for OpenIDM to pick up and synchronize the OpenIDM repository with the remote XML user repository.

Procedure 9.9. Configuring JCS

- Copy the `openidm/samples/sample1/data` directory to a location on the JCS machine.

Procedure 9.10. Configuring OpenIDM for the XML Example

1. Start OpenIDM. You can ignore errors like `cannot connect to database` and `cannot find jdbc driver`. These errors will be fixed once OpenIDM has been configured and restarted.

2. Copy the `openidm/samples/sample1/conf` directory to `openidm/conf`. Overwrite any existing files.

   For Unix/Linux, enter the following in a terminal window.

```
$ cd path/to/openidm
$ cp -r ./samples/sample1/conf ./conf
```

   For Windows, enter the following on the command line.

```
c:\> cd path\to\openidm
.\> copy .\samples\sample1\conf .\conf
```

3. Copy the `openidm/samples/provisioners/provisioner.openicf.connectorinfoprovider.jso` to `openidm/conf`.

```
$ cp . samples/provisioners/provisioner.openicf.connectorinfoprovider.json ./conf
```

4. Edit the `provisioner.openicf.connectorinfoprovider.json` to match your network setup. The following is an example of how it could look.

```
{
     "connectorsLocation" : "connectors",
     "remoteConnectorServers" : [
         {
             "name" : "xml",
             "host" : "10.0.0.2",
             "port" : 8759,
             "useSSL" : false,
             "timeout" : 0,
             "key" : "password"
         }
     ],
 }
```

5. Edit the `provisioner.openicf-xml.json` in the `/conf` directory to read like the following.

```
{
    "name" : "xmlfile",
    "connectorRef" : {
        "connectorHostRef" : "xml",
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
        "bundleName" : "org.forgerock.openicf.connectors.file.openicf-xml-connector",
        "bundleVersion" : "1.1.1.0",
        "connectorName" : "com.forgerock.openicf.xml.XMLConnector"
    },
    "producerBufferSize" : 100,
    "connectorPoolingSupported" : true,
    "poolConfigOption" : {
        "maxObjects" : 10,
        "maxIdle" : 10,
        "maxWait" : 150000,
        "minEvictableIdleTimeMillis" : 120000,
        "minIdle" : 1
    },
    "operationTimeout" : {
        "CREATE" : -1,
        "TEST" : -1,
        "AUTHENTICATE" : -1,
        "SEARCH" : -1,
        "VALIDATE" : -1,
        "GET" : -1,
        "UPDATE" : -1,
        "DELETE" : -1,
        "SCRIPT_ON_CONNECTOR" : -1,
        "SCRIPT_ON_RESOURCE" : -1,
        "SYNC" : -1,
        "SCHEMA" : -1
    },
    "configurationProperties" : {
        "xsdIcfFilePath" : "/Program files (x86)/openicf/data/
        resource-schema-1.xsd",
        "xsdFilePath" : "/Program Files (x86)/openicf/data/
        resource-schema-extension.xsd",
        "xmlFilePath" : "/Program Files (x86)/openicf/data/xmlConnectorData.xml"
    },
    "objectTypes" : {
        "account" : {
            "$schema" : "http://json-schema.org/draft-03/schema",
            "id" : "__ACCOUNT__",
            "type" : "object",
            "nativeType" : "__ACCOUNT__",
            "properties" : {
                "description" : {
                    "type" : "string",
                    "nativeName" : "__DESCRIPTION__",
                    "nativeType" : "string"
                },
                "firstname" : {
                    "type" : "string",
                    "nativeName" : "firstname",
                    "nativeType" : "string"
                },
                "email" : {
                    "type" : "array",
                    "items" : {
                        "type" : "string",
                        "nativeType" : "string"
                    },
                    "nativeName" : "email",
                    "nativeType" : "string"
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
                },
                "__UID__" : {
                    "type" : "string",
                    "nativeName" : "__UID__"
                },
                "password" : {
                    "type" : "string",
                    "required" : false,
                    "nativeName" : "__PASSWORD__",
                    "nativeType" : "JAVA_TYPE_GUARDEDSTRING",
                    "flags" : [
                        "NOT_READABLE",
                        "NOT_RETURNED_BY_DEFAULT"
                    ]
                },
                "name" : {
                    "type" : "string",
                    "required" : true,
                    "nativeName" : "__NAME__",
                    "nativeType" : "string"
                },
                "lastname" : {
                    "type" : "string",
                    "required" : true,
                    "nativeName" : "lastname",
                    "nativeType" : "string"
                }
            }
        }
    },
    "operationOptions" : { }
}
```

6. Verify that the following settings are correct.

   - The value of `connectorHostRef : xml` points to the property `name` of `provisioner.openicf.connectorinfoprovider.json`. This indicates which connectorinfoprovider to use.

   - The `bundleVersion : 1.1.1.0` must be exactly the same as `openicf-scriptedsql-connector-1.1.1.0.jar`. on JCS `/bundles`.

   - The path to `xsdIcfFilePath : /Program files (x86)/openicf/data/resource-schema-1.xs`

7. Restart OpenIDM to verify that all of the configuration changes have occurred. There should be no error message when OpenIDM is restarted. To check, run the following command:

   ```
   src list
   ```

   This returns a list of installed modules, including the following:

   ```
   [  17] [active       ] org.forgerock.openidm.provisioner.openicf
   ```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

> **Note**
>
> The number may differ. Make sure to note the number returned.

When you have installed more connectors, there will be more OpenIFC modules. If the state of the module is active, the module is installed properly. If the state is unsatisfied, then you have not configured it correctly and you must check your configuration. You can also check the content of installed modules. This can be useful if you have an unsatisfied state and you want to check that the content is the same as in the `*.json` file, to verify that the configuration change you made was picked up. To list the content of the module run the following command, with the number returned from the previous step:

```
scr info 17 <your number>
```

> **Note**
>
> You can also check the `provisioner.openicf.connectorinfoprovider`

8. Run reconciliation with the following command.

```
$ curl --header "X-OpenIDM-Username: openidm-admin"
--header "X-OpenIDM-Password: openidm-admin" --request POST
"http://localhost:8080/openidm/recon?_action=recon&mapping=systemXmlfileAccounts_managedUser"
```

This will return a reconciliation id similar to the following:

```
{"_id":"a5346543-db9a-4f8b-ba25-af2a1b576a54"}
```

9. Check the internal repository (OrientDB or MySQL) to make sure that the users were reconciled. For information about connecting to OrientDB, see *Before You Begin* in the *Installation Guide* in the *Installation Guide*. For information about using MySQL as a repository, see *Installing a Repository For Production* in the *Installation Guide* in the *Installation Guide*.

9.4.3.4. Configuring the Active Directory Connector

A sample Active Directory Connector configuration file is provided in `opendim/samples/provisioners/provisioner.openicf-ad.json`. The following excerpt shows the configuration for the connector.

```
{
    "connectorHostRef": "dotnet",
    "connectorName":
        "Org.IdentityConnectors.ActiveDirectory.ActiveDirectoryConnector",
    "bundleName": "ActiveDirectory.Connector",
    "bundleVersion": "1.0.0.6109"
}
```

The `connectorHostRef` must point by name to an existing connector info provider configuration, that you store in `openidm/conf/provisioner.openicf.connectorinfoprovider.json`. The `connectorHostRef` property

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

is required as the Active Directory connector must be installed on a .NET connector server, which is always "remote" relative to OpenIDM.

The following excerpt shows the configuration for the connector info provider.

```
{
    "connectorsLocation": "connectors",
    "remoteConnectorServers": [
        {
            "name": "dotnet",
            "host": "10.0.0.10",
            "port": 8759,
            "useSSL": false,
            "timeout": 0,
            "key": "Passw0rd"
        }
    ]
}
```

The following excerpt shows typical configuration properties.

```
{
    "DirectoryAdminName": "EXAMPLE\\Administrator",
    "DirectoryAdminPassword": "passw0rd",
    "ObjectClass": "User",
    "Container": "dc=example,dc=com",
    "CreateHomeDirectory": true,
    "LDAPHostName": "127.0.0.1",
    "SearchChildDomains": false,
    "DomainName": "example",
    "SyncGlobalCatalogServer": null,
    "SyncDomainController": null,
    "SearchContext": "dc=example,dc=com"
}
```

DirectoryAdminName

Account used to authenticate. This can be a `domainname\user` combination, or simply the user name.

DirectoryAdminPassword

Password used to authenticate

ObjectClass

Object class for user objects

Container

Base context for all searches

CreateHomeDirectory

When `true`, create a home directory for new users.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

LDAPHostName

Use to enforce connection to a particular Active Directory server.

SearchChildDomains

When set to `true` or `false`, apply `SyncGlobalCatalogServer` and `SyncDomainController` settings

DomainName

Windows domain name

SyncGlobalCatalogServer

Global catalog server to use when searching child domains

SyncDomainController

Domain controller to use during synchronization when not searching child domains

SearchContext

Reserved for future use

9.4.3.5. Using PowerShell Scripts With the Active Directory Connector

The Active Directory connector supports PowerShell scripting. The following example shows a simple PowerShell script that is referenced in the connector configuration and can be called over the REST interface.

This PowerShell script creates a new MS SQL user with a username that is specified when the script is called. The script sets the user's password to `Passw0rd` and, optionally, gives the user a role. Save this script as `openidm/script/createUser.ps1`.

```
   if ($loginName -ne $NULL) {
 [System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SMO') | Out-Null
 $sqlSrv = New-Object ('Microsoft.SqlServer.Management.Smo.Server') ('WIN-C2MSQ8G1TCA')

 $login = New-Object -TypeName ('Microsoft.SqlServer.Management.Smo.Login') ($sqlSrv, $loginName)
 $login.LoginType = 'SqlLogin'
 $login.PasswordExpirationEnabled = $false
 $login.Create('Passw0rd')
 #  The next two lines are optional, and to give the new login a server role, optional
 $login.AddToRole('sysadmin')
 $login.Alter()
} else {
 $Error_Message = [string]"Required variables 'loginName' is missing!"
   Write-Error $Error_Message
   throw $Error_Message
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Now edit the Active Directory connector configuration to reference the script. Add the following section to the connector configuration file (`opendim/conf/provisioner.openicf-ad.json`).

```
"systemActions" : [
    {
        "_scriptId" : "ConnectorScriptName",
        "actions" : [
            {
                "systemType" : ".*ActiveDirectoryConnector",
                "actionType" : "Shell",
                "actionSource" : "@echo off \r\n echo %loginName%\r\n"
            },
            {
                "systemType" : ".*ActiveDirectoryConnector",
                "actionType" : "PowerShell",
                "actionFile" : "script/createUser.ps1"
            }
        ]
    }
]
```

To call the PowerShell script over the REST interface, use the following request, specifying the userName as input:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/system/ActiveDirectory/account?
_action=script&_scriptId=ConnectorScriptName&loginName=myUser"
```

9.4.4. CSV File Connector

The CSV file connector often serves when importing users, either for initial provisioning or for ongoing updates. When used continuously in production, a CSV file serves as a change log, often containing only user records that changed.

The following example shows an excerpt of the provisioner configuration. The default connector-jar location is `openidm/connectors`. Therefore the `connectorHostRef` must point to `"#LOCAL"`.

```
{
  "connectorRef": {
    "connectorHostRef": "#LOCAL",
    "connectorName": "org.forgerock.openicf.csvfile.CSVFileConnector",
    "bundleName":
      "org.forgerock.openicf.connectors.csvfile-connector",
    "bundleVersion": "1.1.1.0"
  }
}
```

The following excerpt shows required configuration properties.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
    "configurationProperties": {
        "filePath": "data/hr.csv",
        "uniqueAttribute": "uid"
    }
}
```

The CSV file connector also supports a number of optional configuration properties, in addition to the required properties.

encoding (optional)

Default: `"utf-8"`

fieldDelimiter (optional)

Default: `","`

filePath (required)

References the CSV file containing account entries

multivalueDelimiter (optional)

Used with multi-valued attributes. Default: `";"`

passwordAttribute (optional)

Attribute containing the password. Use when password-based authentication is required.

uniqueAttribute (required)

Primary key used for the CSV file

usingMultivalue (optional)

Whether attributes can have multiple values. Default: `false`

9.4.5. Scripted SQL Connector

The Scripted SQL Connector uses customizable Groovy scripts to interact with the database.

The connector uses one script for each of the following actions on the external database.

- Create
- Delete
- Search
- Sync
- Test

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

- Update

See the openidm/samples/sample3/tools/ directory for example scripts.

The scripted SQL connector runs with autocommit mode enabled by default. As soon as a statement is executed that modifies a table, the update is stored on disk and the change cannot be rolled back. This setting applies to all database actions (search, create, delete, test, synch, and update). You can disable autocommit in the connector configuration file (conf/provisioner.openicf-scriptedsql.json) by adding the autocommit property and setting it to false, for example:

```
"configurationProperties" : {
      "host" : "localhost",
      "port" : "3306",
      ...
      "database" : "HRDB",
      "autoCommit" : false,
      "reloadScriptOnExecution" : true,
      "createScriptFileName" : "/path/to/openidm/tools/CreateScript.groovy",
```

If you require a traditional transaction with a manual commit for a specific script, you can disable autocommit mode in the script or scripts for each action that requires a manual commit.

9.5. Creating Default Connector Configurations

Rather than creating provisioner files by hand, use the service that OpenIDM exposes through the REST interface to create basic connector configuration files named provisioner-openicf-*Connector Name*.json file.

You create a new connector configuration file in three stages.

1. List available connectors.
2. Generate the core configuration.
3. Connect to the target system and generate the final configuration.

List available connectors using the following command.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST "http://localhost:8080/openidm/system?_action=CREATECONFIGURATION"
```

Available connectors are installed in openidm/connectors. OpenIDM bundles the following connectors.

- csvfile
- ldap
- scriptedsql

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

- xml

The command above therefore should return the following output (formatted here with lines folded to make it easier to read.)


```
{
  "connectorRef": [
    {
      "connectorName": "org.identityconnectors.ldap.LdapConnector",
      "bundleName":
        "org.forgerock.openicf.connectors.ldap-connector",
      "bundleVersion": "1.1.1.0"
    },
    {
      "connectorName": "com.forgerock.openicf.xml.XMLConnector",
      "bundleName":
        "org.forgerock.openicf.connectors.file.openicf-xml-connector",
      "bundleVersion": "1.1.1.0"
    },
    {
      "connectorHostRef":
       "osgi:service/org.forgerock.openicf.framework.api.osgi.ConnectorManager",
      "connectorName": "org.forgerock.openicf.connectors.scriptedsql.ScriptedSQLConnector",
      "bundleName":
       "org.forgerock.openicf.connectors.scriptedsql-connector",
      "bundleVersion": "1.1.1.0"
    },
    {
      "connectorHostRef":
       "osgi:service/org.forgerock.openicf.framework.api.osgi.ConnectorManager",
      "connectorName": "org.forgerock.openicf.csvfile.CSVFileConnector",
      "bundleName":
       "org.forgerock.openicf.connectors.csvfile-connector",
      "bundleVersion": "1.1.1.0"
    }
  ]
}
```


To generate the core configuration, choose one of the available connectors by copying JSON objects from the list into the body of the REST command, as shown below for the XML connector.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 -d '{"connectorRef":
 {"connectorName":"com.forgerock.openicf.xml.XMLConnector",
 "bundleName":"org.forgerock.openicf.connectors.file.openicf-xml-connector",
 "bundleVersion":"1.1.1.0"}}'
 --request POST "http://localhost:8080/openidm/system?_action=CREATECONFIGURATION"
```

The command returns a core connector configuration. The core connector configuration returned is not yet functional. It does not contain system specific "configurationProperties" such as the host name and port for web based connectors, or the "xmlFilePath" for the XML file based connectors as can be seen below. In addition, the configuration returned does not include complete "objectTypes" and "operationOptions" parts.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
    "connectorRef": {
        "connectorName": "com.forgerock.openicf.xml.XMLConnector",
        "bundleName":
            "org.forgerock.openicf.connectors.file.openicf-xml-connector",
        "bundleVersion": "1.1.1.0"
    },
    "poolConfigOption": {
        "maxObjects": 10,
        "maxIdle": 10,
        "maxWait": 150000,
        "minEvictableIdleTimeMillis": 120000,
        "minIdle": 1
    },
    "resultsHandlerConfig": {
        "enableNormalizingResultsHandler": true,
        "enableFilteredResultsHandler": true,
        "enableCaseInsensitiveFilter": false,
        "enableAttributesToGetSearchResultsHandler": true
    },
    "operationTimeout": {
        "CREATE": -1,
        "UPDATE": -1,
        "DELETE": -1,
        "TEST": -1,
        "SCRIPT_ON_CONNECTOR": -1,
        "SCRIPT_ON_RESOURCE": -1,
        "GET": -1,
        "RESOLVEUSERNAME": -1,
        "AUTHENTICATE": -1,
        "SEARCH": -1,
        "VALIDATE": -1,
        "SYNC": -1,
        "SCHEMA": -1
    },
    "configurationProperties": {
        "xmlFilePath": null,
        "xsdFilePath": null,
        "xsdIcfFilePath": null
    }
}
```

To generate the final configuration, add the missing "configurationProperties" to the core configuration, and use the updated core configuration as the body for the next command.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --data '{
   "connectorRef" :
      {
         "connectorName" : "com.forgerock.openicf.xml.XMLConnector",
         "bundleName" :
             "org.forgerock.openicf.connectors.file.openicf-xml-connector",
         "bundleVersion" : "1.1.1.0"
      },
   "poolConfigOption" :
      {
         "maxObjects" : 10,
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
            "maxIdle" : 10,
            "maxWait" : 150000,
            "minEvictableIdleTimeMillis" : 120000,
            "minIdle" : 1
         },
      "resultsHandlerConfig" :
         {
            "enableNormalizingResultsHandler" : true,
            "enableFilteredResultsHandler" : true,
            "enableCaseInsensitiveFilter" : false,
            "enableAttributesToGetSearchResultsHandler" : true
         },
      "operationTimeout" :
         {
            "CREATE" : -1,
            "UPDATE" : -1,
            "DELETE" : -1,
            "TEST" : -1,
            "SCRIPT_ON_CONNECTOR" : -1,
            "SCRIPT_ON_RESOURCE" : -1,
            "GET" : -1,
            "RESOLVEUSERNAME" : -1,
            "AUTHENTICATE" : -1,
            "SEARCH" : -1,
            "VALIDATE" : -1,
            "SYNC" : -1,
            "SCHEMA" : -1
         },
      "configurationProperties" :
         {
            "xsdIcfFilePath" : "samples/sample1/data/resource-schema-1.xsd",
            "xsdFilePath" : "samples/sample1/data/resource-schema-extension.xsd",
            "xmlFilePath" : "samples/sample1/data/xmlConnectorData.xml"
         }
 }'
 --request POST "http://localhost:8080/openidm/system?_action=CREATECONFIGURATION"
```

Note

Notice the single quotes around the argument to the `--data` option in the command above. For most UNIX shells, single quotes around a string prevent the shell from executing the command when encountering a newline in the content. You can therefore pass the `--data '...'` option on a single line or including line feeds.

OpenIDM attempts to read the schema, if available, from the external resource in order to generate output. OpenIDM then iterates through schema objects and attributes, creating JSON representations for "objectTypes" and "operationOptions" for supported objects and operations.

```
{
    "connectorRef": {
        "connectorHostRef": "#LOCAL",
        "connectorName": "com.forgerock.openicf.xml.XMLConnector",
        "bundleName":
            "org.forgerock.openicf.connectors.file.openicf-xml-connector",
        "bundleVersion": "1.1.1.0-EA"
    },
    "poolConfigOption": {
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
        "maxObjects": 10,
        "maxIdle": 10,
        "maxWait": 150000,
        "minEvictableIdleTimeMillis": 120000,
        "minIdle": 1
    },
    "resultsHandlerConfig": {
        "enableNormalizingResultsHandler": true,
        "enableFilteredResultsHandler": true,
        "enableCaseInsensitiveFilter": false,
        "enableAttributesToGetSearchResultsHandler": true
    },
    "operationTimeout": {
        "CREATE": -1,
        "UPDATE": -1,
        "DELETE": -1,
        "TEST": -1,
        "SCRIPT_ON_CONNECTOR": -1,
        "SCRIPT_ON_RESOURCE": -1,
        "GET": -1,
        "RESOLVEUSERNAME": -1,
        "AUTHENTICATE": -1,
        "SEARCH": -1,
        "VALIDATE": -1,
        "SYNC": -1,
        "SCHEMA": -1
    },
    "configurationProperties": {
        "xmlFilePath": "samples/sample1/data/xmlConnectorData.xml",
        "xsdFilePath": "samples/sample1/data/resource-schema-extension.xsd",
        "xsdIcfFilePath": "samples/sample1/data/resource-schema-1.xsd"
    },
    "objectTypes": {
        "OrganizationUnit": {
            "...": "..."
        },
        "__GROUP__": {
            "$schema": "http://json-schema.org/draft-03/schema",
            "id": "__GROUP__",
            "type": "object",
            "nativeType": "__GROUP__",
            "properties": {
                "__DESCRIPTION__": {
                    "type": "string",
                    "required": true,
                    "nativeName": "__DESCRIPTION__",
                    "nativeType": "string"
                },
                "__NAME__": {
                    "type": "string",
                    "required": true,
                    "nativeName": "__NAME__",
                    "nativeType": "string"
                }
            }
        },
        "__ACCOUNT__": {
            "$schema": "http://json-schema.org/draft-03/schema",
            "id": "__ACCOUNT__",
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
            "type": "object",
            "nativeType": "__ACCOUNT__",
            "properties": {
                "firstname": {
                    "type": "string",
                    "nativeName": "firstname",
                    "nativeType": "string"
                },
                "__DESCRIPTION__": {
                    "type": "string",
                    "nativeName": "__DESCRIPTION__",
                    "nativeType": "string"
                },
                "__UID__": {
                    "type": "string",
                    "nativeName": "__UID__",
                    "nativeType": "string"
                },
                "__NAME__": {
                    "type": "string",
                    "required": true,
                    "nativeName": "__NAME__",
                    "nativeType": "string"
                }
            }
        }
    },
    "operationOptions": {
        "CREATE": {
            "objectFeatures": {
                "OrganizationUnit": {
                    "...": "..."
                },
                "__GROUP__": {
                    "...": "..."
                },
                "__ACCOUNT__": {
                    "denied": false,
                    "onDeny": "DO_NOTHING",
                    "operationOptionInfo": {
                        "$schema": "http://json-schema.org/draft-03/schema",
                        "id": "FIX_ME",
                        "type": "object",
                        "properties": {
                            "...": "..."
                        }
                    }
                }
            }
        },
        "UPDATE": {
            "objectFeatures": {
                "__ACCOUNT__": {
                    "denied": false,
                    "onDeny": "DO_NOTHING",
                    "operationOptionInfo": {
                        "$schema": "http://json-schema.org/draft-03/schema",
                        "id": "FIX_ME",
                        "type": "object",
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
                        "properties": {
                            "...": "..."
                        }
                    }
                }
            }
        }
    }
}
```

As OpenIDM produces a full property set for all attributes and all object types in the schema from the external resource, the resulting configuration can be large. For an LDAP server, OpenIDM can generate a configuration containing several tens of thousands of lines, for example. You might therefore want to reduce the schema to a minimum on the external resource before you run the final command.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 10
Configuring Synchronization

One of the core services of OpenIDM is synchronizing identity data from different resources. This chapter explains what you must know to get started configuring OpenIDM's flexible synchronization mechanism, and illustrates the concepts with examples.

10.1. Types of Synchronization

Synchronization happens either when OpenIDM receives a change directly, or when OpenIDM discovers a change on an external resource.

For direct changes to OpenIDM, OpenIDM immediately pushes updates to all external resources configured to receive the updates. A direct change can originate not only as a write request through the REST interface, but also as an update resulting from reconciliation with another resource.

OpenIDM discovers changes on external resources through reconciliation and through LiveSync.

Reconciliation

In identity management, *reconciliation* is the process of bidirectional synchronization of objects between different data stores. Reconciliation applies mainly to user objects, although OpenIDM can reconcile any objects, including groups and roles.

To perform reconciliation, OpenIDM analyzes both source and target systems to uncover the differences that it must reconcile. Reconciliation can therefore be a heavyweight process. When working with large data sets, finding all changes can be more work than processing the changes.

Reconciliation is, however, thorough. It recognizes system error conditions and catches changes that might be missed by the more lightweight LiveSync mechanism. Reconciliation therefore serves as the basis for compliance and reporting functionality.

LiveSync

LiveSync performs the same job as reconciliation. LiveSync relies on a change log on the external resource to determine which objects have changed.

LiveSync is intended to react quickly to changes as they happen. LiveSync is however a best effort mechanism that, in some cases, can miss changes.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Furthermore, not all resources provide the change log mechanism that LiveSync requires. The change log provides OpenIDM with a list of objects that have changed since the last request, so that OpenIDM does not need to scan all objects for changes. OpenDJ and Active Directory both provide the change log required for LiveSync.

To determine what to synchronize, and how to carry out synchronization, OpenIDM relies on mappings that you configure. LiveSync relies on the set of mappings that you configure once per OpenIDM server. Reconciliation allows you to configure specific mappings for a particular reconciliation job.

You must trigger OpenIDM to poll for changes on external resources, usually by scheduling reconciliation or LiveSync, as described in *Scheduling Tasks and Events*. Alternatively, you can manage reconciliation and LiveSync over the REST interface, as described in the following sections.

10.2. Managing Reconciliation Over REST

You can trigger, cancel, and monitor reconciliation operations over REST, using the REST endpoint `http://localhost:8080/openidm/recon`.

10.2.1. Triggering a Reconciliation Run

The following example triggers a reconciliation operation based on the `systemLdapAccounts_managedUser` mapping. The mapping is defined in the file `conf/sync.json`.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/recon?_action=recon&mapping=systemLdapAccounts_managedUser"
```

By default, an assigned reconciliation run ID is returned immediately when the reconciliation operation is initiated. Clients can make subsequent calls to the reconciliation service, using this reconciliation run ID to query its state and to call operations on it.

For example, the reconciliation run initiated previously would return something similar to the following:

```
{"_id":"0890ad62-4738-4a3f-8b8e-f3c83bbf212e"}
```

To have the entire reconciliation run complete before the reconciliation run ID is returned, set the `waitForCompletion` property to `true` when the reconciliation is initiated. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/recon?_action=recon&
 mapping=systemLdapAccounts_managedUser&waitForCompletion=true"
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

10.2.2. Canceling a Reconciliation Run

You can cancel a reconciliation run by sending a REST call with the `cancel` action, specifying the reconciliation run ID. For example, the following call cancels the reconciliation run initiated in the previous section:

```
$curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/recon/0890ad62-4738-4a3f-8b8e-f3c83bbf212e?_action=cancel"
```

The output for a reconciliation cancelation request is similar to the following:

```
    {"_id":"0890ad62-4738-4a3f-8b8e-f3c83bbf212e",
     "action":"cancel",
     "status":"SUCCESS"}
```

If you specified that the call should wait for completion before the ID is returned, you can obtain the reconciliation run ID from the list of active reconciliations, as described in the following section.

10.2.3. Listing Reconciliation Runs

You can display a list of reconciliation processes that have completed, and those that are in progress, by running a RESTful GET on `"http://localhost:8080/openidm/recon"`. The following example displays all reconciliation runs.

```
$curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/recon"
```

The output of such a request is similar to the following, with one item for each reconciliation run.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{"reconciliations":[
 {"_id":"d3040cc9-ec2e-41b8-86c4-72393087a626",
 "mapping":"systemLdapAccounts_managedUser",
 "state":"SUCCESS",
 "stage":"COMPLETED_SUCCESS",
 "stageDescription":"reconciliation completed.",
 "progress":{
   "source":{
      "existing":{
         "processed":1001,
         "total":"1001"
      }
   },
   "target":{
      "existing":{
         "processed":1001,
         "total":"1001"
      },
      "created":0
   },
   "links":{
      "existing":{
         "processed":1001,
         "total":"1001"
      },
      "created":0
   }
 },
 "started":"2012-11-18T08:48:00.031Z",
 "ended":"2012-11-18T08:48:00.160Z"},
```


Each reconciliation run has the following properties:

`_id`

The ID of the reconciliation run.

`mapping`

The name of the mapping, defined in the `conf/sync.json` file.

`state`

The high level state of the reconciliation run. Values can be as follows:

- `ACTIVE`

  The reconciliation run is in progress.

- `CANCELED`

  The reconciliation run was successfully canceled.

- `FAILED`

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The reconciliation run was terminated because of failure.

- `SUCCESS`

  The reconciliation run completed successfully.

`stage`

The current stage of the reconciliation run's progress. Values can be as follows:

- `ACTIVE_INITIALIZED`

  The initial stage, when a reconciliation run is first created.

- `ACTIVE_QUERY_ENTRIES`

  Querying the source, target and possibly link sets to reconcile.

- `ACTIVE_RECONCILING_SOURCE`

  Reconciling the set of IDs retrieved from the mapping source.

- `ACTIVE_RECONCILING_TARGET`

  Reconciling any remaining entries from the set of IDs retrieved from the mapping target, that were not matched or processed during the source phase.

- `ACTIVE_LINK_CLEANUP`

  Checking whether any links are now unused and should be cleaned up.

- `ACTIVE_PROCESSING_RESULTS`

  Post-processing of reconciliation results.

- `ACTIVE_CANCELING`

  Attempting to abort a reconciliation run in progress.

- `COMPLETED_SUCCESS`

  Successfully completed processing the reconciliation run.

- `COMPLETED_CANCELED`

  Completed processing because the reconciliation run was aborted.

- `COMPLETED_FAILED`

  Completed processing because of a failure.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`stageDescription`

A description of the stages described previously.

`progress`

The progress object has the following structure (annotated here with comments):

```
"progress":{
      "source":{             // Progress on the set of existing entries in the mapping source
         "existing":{
            "processed":1001,
            "total":"1001"   // Total number of entries in source set, if known, “?” otherwise
         }
      },
      "target":{             // Progress on the set of existing entries in the mapping target
         "existing":{
            "processed":1001,
            "total":"1001"   // Total number of entries in target set, if known, “?” otherwise
         },
         "created":0         // New entries that were created
      },
      "links":{              // Progress on the set of existing links between source and target
         "existing":{
            "processed":1001,
            "total":"1001"   // Total number of existing links, if known, “?” otherwise
         },
         "created":0         // Denotes new links that were created
      }
},
```

10.3. Triggering LiveSync Over REST

The ability to trigger LiveSync operations over REST, or by using the resource API, enables you to use an external scheduler to trigger a LiveSync operation, rather than using the OpenIDM scheduling mechanism.

There are two ways in which to trigger a LiveSync operation over REST.

- Use the `_action=liveSync` parameter directly on the resource. This is the recommended method. The following example calls a LiveSync operation on the user accounts in an external LDAP system.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/system/ldap/account?_action=liveSync"
```

- Target the `system` endpoint and supply a `source` parameter to identify the object that should be synchronized. This method matches the scheduler configuration and can therefore be used to test schedules before they are implemented.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The following example calls the same LiveSync operation as the previous example.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/system?_action=liveSync&source=system/ldap/account"
```

A successful LiveSync operation returns the following response:

```
{
    "_id": "SYSTEMLDAPACCOUNT",
    "_rev": "0",
    "connectorData": {
        "syncToken": 1,
        "nativeType": "integer"
    }
}
```

You should not run two identical LiveSync operations simultaneously - you must ensure that the first operation has completed before a second similar operation is launched.

To troubleshoot a LiveSync operation that has not succeeded, you can include an optional parameter (`detailedFailure`) to return additional information. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/system/ldap/account?_action=liveSync&detailedFailure=true"
```

Note

The first time that a LiveSync operation is called, no synchronization token exists in the database to establish which changes have already been processed. The default LiveSync behavior is to locate the last existing entry in the change log, and to store that entry in the database as the current starting position from which changes should be applied. This behavior prevents LiveSync from processing changes that might already have been processed during an initial data load. Subsequent LiveSync operations will pick up and process any new changes.

Typically, in setting up LiveSync on a new system, you would load the data initially (by using reconciliation, for example) and then enable LiveSync, starting from that base point.

10.4. Flexible Data Model

Identity management software tends to favor either a meta-directory data model, where all data are mirrored in a central repository, or a virtual data model, where only a minimum set of attributes are stored centrally, and most are loaded on demand from the external resources in which they are

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

stored. The meta-directory model offers fast access at the risk of getting out-of-date data. The virtual model guarantees fresh data, but pays for that guarantee in terms of performance.

OpenIDM leaves the data model choice up to you. You determine the right trade offs for a particular deployment. OpenIDM does not hard code any particular schema or set of attributes stored in the repository. Instead, you define how external system objects map onto managed objects, and OpenIDM dynamically updates the repository to store the managed object attributes that you configure.

You can, for example, choose to follow the data model defined in the Simple Cloud Identity Management (SCIM) specification. The following object represents a SCIM user.

```
{
    "userName": "james1",
    "familyName": "Berg",
    "givenName": "James",
    "email": [
        "james1@example.com"
    ],
    "description": "Created by OpenIDM REST.",
    "password": "asdfkj23",
    "displayName": "James Berg",
    "phoneNumber": "12345",
    "employeeNumber": "12345",
    "userType": "Contractor",
    "title": "Vice President",
    "active": true
}
```

Note

Avoid using the dash character (`-`) in property names, like `last-name`, as dashes in names make JavaScript syntax more complex. If you cannot avoid the dash, then write `source['last-name']` instead of `source.last-name` in java script.

10.5. Basic Data Flow Configuration

Data flow for synchronization involves the following elements:

- Connector configuration files (`conf/provisioner-*.json`), with one file per external resource.
- Synchronization mappings file (`conf/sync.json`), with one file per OpenIDM instance.
- A links table that OpenIDM maintains in its repository.
- The scripts required to check objects and manipulate attributes.

10.5.1. Connector Configuration Files

Connector configuration files map external resource objects to OpenIDM objects, and are described in detail in the chapter on *Connecting to External Resources*. Connector configuration files are

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

named `openidm/conf/provisioner.resource-name.json`, where *resource-name* reflects the connector technology and external resource, such as `openicf-xml`.

An excerpt from an example connector configuration follows. The example shows the name for the connector and two attributes of an account object type. In the attribute mapping definitions, the attribute name is mapped from the `nativeName`, the attribute name used on the external resource, to the attribute name used in OpenIDM. Thus the example shows that the `sn` attribute in LDAP is mapped to `lastName` in OpenIDM. The `homePhone` attribute can have multiple values.

```
{
    "name": "MyLDAP",
    "objectTypes": {
        "account": {
            "lastName": {
                "type": "string",
                "required": true,
                "nativeName": "sn",
                "nativeType": "string"
            },
            "homePhone": {
                "type": "array",
                "items": {
                    "type": "string",
                    "nativeType": "string"
                },
                "nativeName": "homePhone",
                "nativeType": "string"
            }
        }
    }
}
```

In order for OpenIDM to access external resource objects and attributes, the object and its attributes must match the connector configuration. Note that the connector file only maps external resource objects to OpenIDM objects. To construct attributes and to manipulate their values, you use the synchronization mappings file.

10.5.2. Synchronization Mappings File

The synchronization mappings file (`openidm/conf/sync.json`) represents the core configuration for OpenIDM synchronization.

The `sync.json` file describes a set of mappings. Each mapping specifies how attributes from source objects correspond to attributes on target objects. The source and target indicate the direction for the data flow, so you must define a separate mapping for each data flow. For example, if you want data flows from an LDAP server to the repository and also from the repository to the LDAP server, you must define two separate mappings.

You identify external resource sources and targets as `system/name/object-type`, where *name* is the name used in the connector configuration file, and *object-type* is the object defined in the connector configuration file list of object types. For objects in OpenIDM's internal repository, you use

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`managed/object-type`, where *object-type* is defined in `openidm/conf/managed.json`. The name for the mapping by convention is set to a string of the form `source_target`, as shown in the following example.

```
{
    "mappings": [
        {
            "name": "systemLdapAccounts_managedUser",
            "source": "system/MyLDAP/account",
            "target": "managed/user",
            "properties": [
                {
                    "target": "familyName",
                    "source": "lastName"
                },
                {
                    "target": "homePhone",
                    "source": "homePhone"
                },
                {
                    "target": "phoneExtension",
                    "default": "0047"
                },
                {
                    "target": "mail",
                    "comment": "Set mail if non-empty.",
                    "source": "email",
                    "condition": {
                        "type": "text/javascript",
                        "source": "(object.email != null)"
                    }
                },
                {
                    "target": "displayName",
                    "source": "";
                    "transform": {
                        "type": "text/javascript",
                        "source": "(source.lastName +', ' + source.firstName;)"
                    }
                }
            ]
        }
    ]
}
```

In this example, the source is the external resource, `MyLDAP`, and the target is OpenIDM's repository, specifically the managed user objects. The `properties` reflect OpenIDM attribute names. For example, the mapping has the attribute `lastName` defined in the `MyLDAP` connector configuration file mapped to `familyName` in the OpenIDM managed user object. Notice that the attribute names come from the connector configuration, rather than the external resource itself.

You can create attributes on the target as part of the mapping. In the example, OpenIDM creates a `phoneExtension` attribute with a default value of `0047`.

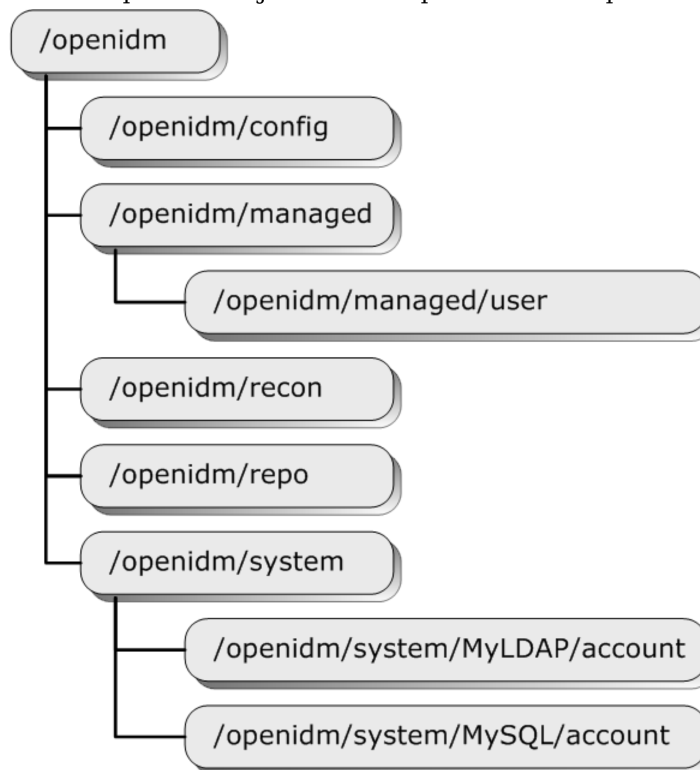
Copyright © 2011-2017 ForgeRock AS. All rights reserved.

You can also set up conditions under which OpenIDM maps attributes as shown for the email attribute in the example. By default, OpenIDM synchronizes all attributes. In the example, the mail attribute is set only if the script for the condition returns `true`.

OpenIDM also enables you to transform attributes. In the example, the value of the `displayName` attribute is set using a combination of the `lastName` and `firstName` attribute values from the source. For transformations, the `source` property is optional. However, the source object is only available when you specify the `source` property. Therefore, in order to use `source.lastName` and `source.firstName` to calculate the `displayName`, the example specifies `"source" : ""`.

To add a flow from the repository to `MyLDAP`, you would define a mapping with source `managed/user` and target `system/MyLDAP/account`, named for example `managedUser_systemLdapAccounts`.

The following image shows the paths to objects in the OpenIDM namespace.



OpenIDM stores managed objects in the repository, and exposes them under `/openidm/managed`. System objects on external resources are exposed under `/openidm/system`.

By default, OpenIDM synchronizes all objects that match those defined in the connector configuration for the resource. Many connectors allow you to limit the scope of objects that the connector accesses. For example, the LDAP connector allows you to specify base DNs and LDAP filters so that you do not need to access every entry in the directory. OpenIDM also allows you to filter what is considered a

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

valid source or valid target for synchronization by using JavaScript code. To apply these filters, use the `validSource`, and `validTarget` properties in your mapping.

validSource

A script that determines if a source object is valid to be mapped. The script yields a boolean value: `true` indicates that the source object is valid; `false` can be used to defer mapping until some condition is met. In the root scope, the source object is provided in the `"source"` property. If the script is not specified, then all source objects are considered valid.

```
{
    "validSource": {
        "type": "text/javascript",
        "source": "source.ldapPassword != null"
    }
}
```

validTarget

A script, used during reconciliation's second phase, that determines if a target object is valid to be mapped. The script yields a boolean value: `true` indicates that the target object is valid; `false` indicates that the target object should not be included in reconciliation. In the root scope, the source object is provided in the `"target"` property. If the script is not specified, then all target objects are considered valid for mapping.

```
{
    "validTarget": {
        "type": "text/javascript",
        "source": "target.employeeType == 'internal'"
    }
}
```

During synchronization, your scripts always have access to a `source` object and a `target` object. Examples already shown in this section use `source.attributeName` to retrieve attributes from the source objects. Your scripts can also write to target attributes using `target.attributeName` syntax.

```
{
    "onUpdate": {
        "type": "text/javascript",
        "source": "if ((source.email != null) {target.mail = source.email;}"
    }
}
```

See the *Scripting Reference* appendix for more on scripting.

By default, all mappings participate in automatic synchronization operations. You can prevent a specific mapping from participating in automatic synchronization by setting the `"enableSync"` property of that mapping to false. In the following example, automatic synchronization is disabled. This means that changes to objects in the LDAP directory are not automatically propagated to the internal repository. To propagate the changes to the internal repository, reconciliation must be launched manually.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
    "mappings" : [
        {
            "name" : "systemLdapAccounts_managedUser",
            "source" : "system/ldap/account",
            "target" : "managed/user",
            "enableSync" : false,
             ....
}
```

10.5.3. Using Encrypted Values

OpenIDM supports reversible encryption of attribute values for managed objects. Attribute values to encrypt include passwords, authentication questions, credit card numbers, and social security numbers. If passwords are already encrypted on the external resource, they are generally excluded from the synchronization process. For more information, see *Managing Passwords*.

You configure encryption in the managed object configuration (in the `openidm/conf/managed.json` file). The following example shows a managed object configuration that encrypts and decrypts `securityAnswer`, `ssn`, and `password` attributes using the default symmetric key, and additional scripts for extra passwords.

```
{
    "objects": [
        {
            "name": "user",
            "properties": [
                {
                    "name": "securityAnswer",
                    "encryption": {
                        "key": "openidm-sym-default"
                    }
                },
                {
                    "name": "ssn",
                    "encryption": {
                        "key": "openidm-sym-default"
                    }
                },
                {
                    "name": "password",
                    "encryption": {
                        "key": "openidm-sym-default"
                    }
                }
            ],
            "onStore": {
                "type": "text/javascript",
                "file": "script/encryptExtraPassword.js"
            },
            "onRetrieve": {
                "type": "text/javascript",
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
                "file": "script/decryptExtraPassword.js"
            }
        }
    ]
}
```

Do not use the default symmetric key, `openidm-sym-default`, in production. See the chapter on *Securing and Hardening OpenIDM* for instructions on adding your own symmetric key.

10.5.4. Restricting HTTP Access to Sensitive Data

You can protect specific sensitive data stored in the repository by marking the corresponding properties as "private". Private data, whether it is encrypted or not, is not accessible over the REST interface. Properties that are marked as private are removed from an object when that object is retrieved over REST.

To mark a property as private, set its `"scope"` to `"private"` in the `conf/managed.json` file.

The following extract of the `managed.json` file shows how HTTP access is prevented on the `password` and `securityAnswer` properties.

```
        "properties" : [
            {
                "name" : "securityAnswer",
                "encryption" : {
                    "key" : "openidm-sym-default"
                },
                "scope" : "private"
            },
            {
                "name" : "password",
                "encryption" : {
                    "key" : "openidm-sym-default"
                },
                "scope" : "private"
```

A potential caveat with using private properties is that such properties are *removed* if an object is updated by using an HTTP `PUT` request. A `PUT` request replaces the entire object in the repository. Because properties that are marked as private are ignored in HTTP requests, these properties are effectively removed from the object when the update is done. To work around this limitation, do not use `PUT` requests if you have configured private properties. Instead, use a `PATCH` request to update only those properties that need to be changed.

For example, to update the familyName of user joe, replace only the familyName and not the entire user object, as follows:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --header "Content-Type: application/json"
 --request POST
 --data '[{
          "replace":"familyName","value": "Brown"
         }]'
 "http://localhost:8080/openidm/managed/user?_action=patch&_queryId=for-userName&uid=joe"
```

Note

The filtering of private data applies only to direct HTTP read and query calls on managed objects. No automatic filtering is done for internal callers, and the data that these callers choose to expose.

10.5.5. Constructing and Manipulating Attributes

OpenIDM enables you to construct and manipulate attributes using scripts that are triggered when an object is created (onCreate), updated (onUpdate), or deleted (onDelete), or when a link is created (onLink), or removed (onUnlink).

The following example derives a DN for an LDAP entry when the entry is created in the internal repository.

```
{
    "onCreate": {
        "type": "text/javascript",
        "source":
            "target.dn = 'uid=' + source.uid + ',ou=people,dc=example,dc=com'"
    }
}
```

10.5.6. Reusing Links

When two mappings exist to synchronize the same objects bidirectionally, you can use the `links` property in one mapping to have OpenIDM use the same internally managed link for both mappings. Otherwise, if no `links` property is specified, OpenIDM maintains a link for each mapping.

The following excerpt shows two mappings, one from MyLDAP accounts to managed users, and another from managed users to MyLDAP accounts. In the second mapping, the `link` property tells OpenIDM to reuse the links created in the first mapping, rather than create new links.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
    "mappings": [
        {
            "name": "systemMyLDAPAccounts_managedUser",
            "source": "system/MyLDAP/account",
            "target": "managed/user"
        },
        {
            "name": "managedUser_systemMyLDAPAccounts",
            "source": "managed/user",
            "target": "system/MyLDAP/account",
            "links": "systemMyLDAPAccounts_managedUser"
        }
    ]
}
```

10.6. Synchronization Situations and Actions

During synchronization, OpenIDM categorizes objects by situation. Situations are characterized by whether an object exists on a source or target system, whether OpenIDM has registered a link between the source object and the target object, and whether the object is considered valid. OpenIDM takes action depending on the situation.

You can define actions for particular situations in the `policies` section of a synchronization mapping, as shown in the following excerpt.

```
{
    "policies": [
        {
            "situation": "CONFIRMED",
            "action": "UPDATE"
        },
        {
            "situation": "FOUND",
            "action": "IGNORE"
        },
        {
            "situation": "ABSENT",
            "action": "CREATE"
        },
        {
            "situation": "AMBIGUOUS",
            "action": "IGNORE"
        },
        {
            "situation": "MISSING",
            "action": "IGNORE"
        },
        {
            "situation": "UNQUALIFIED",
            "action": "IGNORE"
        },
        {
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
            "situation": "UNASSIGNED",
            "action": "IGNORE"
        }
    ]
}
```

If you do not define a policy for a particular situation, OpenIDM takes the default action for the situation.

The situations and their corresponding actions are described in the following sections.

10.6.1. Synchronization Situations

OpenIDM performs a reconciliation action in two phases:

1. Source reconciliation, where OpenIDM accounts for source objects and associated links based on the configured mapping.
2. Target reconciliation, where OpenIDM iterates over the target objects that were not processed in the first phase.

During reconciliation OpenIDM builds three lists, assigning values to the objects to reconcile.

1. All valid objects from the source

   OpenIDM assigns valid source objects `qualifies=1`. Invalid objects, including those not found in the source system, and those filtered out by the script specified `validSource` property, are assigned `qualifies=0`.

2. All records from the appropriate link table

   Objects with corresponding links in the link table of the repository are assigned `link=1`. Objects without corresponding links are assigned `link=0`.

3. All valid objects on the target system

   Objects found in the target system are assigned `target=1`. Objects that are not found in the target system are assigned `target=0`.

Based on the values assigned to objects during source reconciliation, OpenIDM assigns situations, listed here with their default actions.

"CONFIRMED" (qualifies=1, link=1, target=1)

The mapping qualifies for a target object, and a link to an existing target object was found. Detected during change events and reconciliation. Default action: "UPDATE".

"FOUND" (qualifies=1, link=0, target=1)

The mapping qualifies for a target object, there is no link to a target object, and there is a single target object, correlated by the logic found in the correlationQuery. Detected during change events and reconciliation. Default action: "UPDATE".

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

"ABSENT" (qualifies=1, link=0, target=0)

The mapping qualifies for a target object, there is no link to a target object, and there is no correlated target object found. Detected during change events and reconciliation. Default action: "CREATE".

"AMBIGUOUS" (qualifies=1, link=0, target>1)

The mapping qualifies for a target object, there is no link to a target object, but there is more than one correlated target object. Detected during source object changes and reconciliation. Default action: "EXCEPTION".

"MISSING" (qualifies=1, link=1, target=0)

The mapping qualifies for a target object, and there is a qualified link to a target object, but the target object is missing. Only detected during reconciliation and source object changes in synchronous mappings. Default action: "EXCEPTION".

"UNQUALIFIED" (qualifies=0, link=0 or 1, target=1 or >1)

The mapping is not qualified for a source object. One or more targets are found through the correlation logic. Detected during change events and reconciliation. Default action: "DELETE".

"TARGET_IGNORED" (qualifies=0, link=0 or 1, target=1)

The mapping is not qualified for a source object. One or more targets are found through the correlation logic. Detected only during source object changes. Default action: "IGNORE".

"SOURCE_IGNORED" (qualifies=0, link=0, target=0)

The source object is unqualified (by validSource), no link, no target is found. Detected during source object changes and reconciliation. Default action: "IGNORE".

"LINK_ONLY" (qualifies=n/a, link=1, target=0)

The source may or may not be qualified, a link is found, but no target object is found. Detected only during source object changes. Default action: "EXCEPTION".

"ALL_GONE" (qualifies=n/a, link=0, no previous object available)

The source may or may not be qualified, no link is found, and the server is unable to correlate, as the source cannot supply the deleted object. Detected only during source object changes. Default action: "IGNORE".

Based on the values assigned to objects during target reconciliation, OpenIDM assigns situations, listed here with their default actions.

"TARGET_IGNORED" (qualifies=0)

During target reconciliation the target becomes unqualified by the "validTarget" script. Only detected during reconciliation. Default action: "IGNORE"

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

"UNASSIGNED" (qualifies=1, link=0)

A target object exists for which there is no link. Only detected during reconciliation. Default action: "EXCEPTION".

"CONFIRMED" (qualifies=1, link=1, source=1)

The mapping qualifies for a target object, and a link to a source object exists. Detected only during reconciliation. Default action: "UPDATE".

"UNQUALIFIED" (qualifies=0, link=1, source=1, but source does not qualify)

The mapping is not qualified (by validTarget) for a target object, and there is a link from an existing source object where the source exists. Detected during change events and reconciliation. Default action: "DELETE".

SOURCE_MISSING (qualifies=1, link=1, source=0)

The target qualifies and a link is found. But the source object is missing. Detected during change events and reconciliation. Default action: "DELETE".

The following sections reiterate in detail how OpenIDM assigns situations during each of the two synchronization phases.

10.6.2. Source Reconciliation

OpenIDM starts reconciliation and LiveSync by reading a list of objects from the resource. For reconciliation, the list includes all objects available through the connector. For LiveSync, the list contains only changed objects. The connector can filter objects out of the list, too. You can filter objects out of the list by using the `validSource` property.

OpenIDM then iterates over the list, checking each entry against the `validSource` filter, classifying objects according to their situations as described in Section 10.6.1, "Synchronization Situations". OpenIDM uses the list of links for the current mapping to classify objects. Finally, OpenIDM executes the action configured for the situation.

The following table shows how OpenIDM assigns the appropriate situation during source reconciliation, depending on whether a valid source exists (Source Qualifies), whether a link with the appropriate type exists in the repository (Link Exists), and how many target objects are found, either based on links or correlation results.

Table 10.1. Resolving Source Reconciliation Situations

Source Qualifies?		Link Exists?		Target Objects Found[a]			Situation
Yes	No	Yes	No	0	1	> 1	
	X				X		TARGET_IGNORED

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Source Qualifies?		Link Exists?		Target Objects Found[a]			Situation
Yes	No	Yes	No	0	1	> 1	
	X		X		X		SOURCE_MISSING
	X		X			X	UNQUALIFIED
	X	X		X			UNQUALIFIED
	X	X			X		TARGET_IGNORED
	X	X				X	UNQUALIFIED
X			X	X			ABSENT
X			X		X		FOUND
X			X			X	AMBIGUOUS
X		X		X			MISSING
X		X			X		CONFIRMED

[a] If no link exists for the source object, then OpenIDM executes a correlation query. If no previous object is available, OpenIDM cannot correlate.

10.6.3. Target Reconciliation

During source reconciliation, OpenIDM cannot detect situations where no source object exists, such as the UNASSIGNED situation. When no source object exists, OpenIDM detects the situation during the second reconciliation phase, target reconciliation. During target reconciliation, OpenIDM iterates over all target objects that do not have a representation on the source, checking each object against the `validTarget` filter, determining the appropriate situation, and executing the action configured for the situation.

The following table shows how OpenIDM assigns the appropriate situation during target reconciliation, depending on whether a valid target exists (Target Qualifies), whether a link with an appropriate type exists in the repository (Link Exists), whether a source object exists (Source Exists), and whether the source object qualifies (Source Qualifies). Not all situations assigned during source reconciliation are assigned during target reconciliation.

Table 10.2. Resolving Target Reconciliation Situations

Target Qualifies?		Link Exists?		Source Exists?		Source Qualifies?		Situation
Yes	No	Yes	No	Yes	No	Yes	No	
	X							TARGET_IGNORED
X			X		X			UNASSIGNED
X		X		X		X		CONFIRMED
X		X		X			X	UNQUALIFIED
X		X			X			SOURCE_MISSING

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

10.6.4. Situations Specific to Automatic Synchronization and LiveSync

Certain situations occur only during automatic synchronization (when OpenIDM pushes changes made in the repository out to external systems) and LiveSync (when OpenIDM polls external system change logs for changes and updates the repository).

The following table shows the situations that pertain only to automatic sync and LiveSync, when records are *deleted* from the source or target resource.

Table 10.3. Resolving Automatic Sync and LiveSync Delete Situations

Source Qualifies?		**Link Exists?**		**Target Objects Found**[a]			**Situation**
Yes	**No**	**Yes**	**No**	**0**	**1**	**> 1**	
N/A	N/A	X		X			LINK_ONLY
N/A	N/A		X	X			ALL_GONE
X			X			X	AMBIGUOUS
	X		X			X	UNQUALIFIED

[a]If no link exists for the source object, then OpenIDM executes a correlation query. If no previous object is available, OpenIDM cannot correlate.

10.6.5. Synchronization Actions

Once OpenIDM has assigned a situation to an object, OpenIDM takes the actions configured in the mapping. If no action is configured, then OpenIDM takes the default action for the situation. OpenIDM supports the following actions.

"CREATE"

Create and link a target object.

"UPDATE"

Link and update a target object.

"DELETE"

Delete and unlink the target object.

"LINK"

Link the correlated target object.

"UNLINK"

Unlink the linked target object.

"EXCEPTION"

Flag the link situation as an exception.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Do *not* use this action for LiveSync mappings.

"IGNORE"

Do not change the link or target object state.

10.6.6. Providing a Script as an Action

In addition to the static synchronization actions described in the previous section, you can provide a script that is run in specific synchronization situations. The following extract of a sample `sync.json` file specifies that when a synchronization operation assesses an entry as `ABSENT`, the workflow named `managedUserApproval` is invoked. The parameters for the workflow are passed in as properties of the `action` parameter.

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

The variables available to these scripts are described in *Variables Available in Scripts* in the *Scripting Appendix*.

10.7. Asynchronous Reconciliation

Reconciliation can work in tandem with workflows to provide additional business logic to the reconciliation process. You can define scripts to determine the action that should be taken for a particular reconciliation situation. A reconciliation process can launch a workflow after it has assessed a situation, and then perform the reconciliation, or some other action.

For example, you might want a reconciliation process to assess new user accounts that need to be created on a target resource. However, new user account creation might require some kind of approval from a manager before the accounts are actually created. The initial reconciliation process can assess the accounts that need to be created, launch a workflow to request management approval for those accounts, and then relaunch the reconciliation process to create the accounts, once the management approval has been received.

In this scenario, the defined script returns `IGNORE` for new accounts and the reconciliation engine does not continue processing the given object. The script then initiates an asynchronous process which calls back and completes the reconciliation process at a later stage.

A sample configuration for this scenario is available in `openidm/samples/sample9`, and described in *Sample 9 - Asynchronous Reconciliation Using Workflows* in the *Installation Guide* in the *Installation Guide*.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Configuring asynchronous reconciliation involves the following steps:

1. Create the workflow definition file (`.bar` file) and place it in the `openidm/workflow` directory. For more information about creating workflows, see *Integrating Business Processes and Workflows*.

2. Modify the `conf/sync.json` file for the situation or situations that should call the workflow. Reference the workflow name in the configuration for that situation.

   For example, the following `sync.json` extract calls the `managedUserApproval` workflow if the situation is assessed as `ABSENT`:

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

3. In the sample configuration, the workflow calls a second, asynchronous reconciliation process as a final step.

10.8. Configuring Case Sensitivity for Data Stores

By default, OpenIDM is case sensitive, which means that case is taken into account when comparing IDs during reconciliation. For data stores that are case insensitive, such as OpenDJ, IDs and links that are created by a reconciliation process may be stored with a different case to the way in which they are stored in the OpenIDM repository. Such a situation can cause problems during a reconciliation operation, as the links for these IDs may not match.

For such data stores, you can configure OpenIDM to ignore case during reconciliation operations. With case sensitivity turned off in OpenIDM, for those specific mappings, comparisons are done without regard to case.

To specify that data stores are not case sensitive, set the `"sourceIdsCaseSensitive"` or `"targetIdsCaseSensitive"` property to `false` in the mapping for those links. For example, if the LDAP data store is case insensitive, set the mapping from the LDAP store to the managed user repository as follows:

```
"mappings" : [
{
"name" : "systemLdapAccounts_managedUser",
"source" : "system/ldap/account",
"sourceIdsCaseSensitive" : false,
"target" : "managed/user",
"properties" : [
...
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

If a mapping inherits links by using the `"links"` property, it is not necessary to set case sensitivity, because the mapping uses the setting of the referred links.

10.9. Reconciliation Optimization

By default, reconciliation is configured to function in an optimized way. Some of these optimizations might, however, be unsuitable for your environment. The following sections describe the optimizations and how they can be configured.

10.9.1. Correlating Empty Target Sets

To optimize a reconciliation operation, the reconciliation process does not attempt to correlate source objects to target objects if the set of target objects is empty when the correlation is started. This considerably speeds up the process the first time the reconciliation is run. You can change this behavior for a specific mapping by adding the `correlateEmptyTargetSet` property to the mapping definition and setting it to `true`. For example:

```
{
    "mappings": [
        {
            "name"                     : "systemMyLDAPAccounts_managedUser",
            "source"                   : "system/MyLDAP/account",
            "target"                   : "managed/user",
            "correlateEmptyTargetSet"  : true
        },
    ]
}
```

Be aware that this setting will have a performance impact on the reconciliation process.

10.9.2. Prefetching Links

All links are queried at the start of a correlation and the results of that query are used. You can disable the prefetching of links, so that the correlation process looks up each link in the database as it processes each source or target object. You can disable the prefetching of links by adding the `prefetchLinks` property to the mapping, and setting it to `false`, for example:

```
{
    "mappings": [
        {
            "name": "systemMyLDAPAccounts_managedUser",
            "source": "system/MyLDAP/account",
            "target": "managed/user"
            "prefetchLinks" : false
        }
    ]
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Be aware that this setting will have a performance impact on the reconciliation process.

10.9.3. Parallel Reconciliation Threads

By default, reconciliation is executed in a multi-threaded manner, that is, numerous threads are dedicated to the same reconciliation run. Multithreading generally improves reconciliation run performance. The default number of threads for a single reconciliation run is ten (plus the main reconciliation thread). Under normal circumstances, you should not need to change this number, however the default might not be appropriate in the following situations:

- The hardware has many cores and supports more concurrent threads. As a rule of thumb for performance tuning, start with setting the thread number to two times the number of cores.
- The source or target is an external system with high latency or slow response times. Threads may then spend considerable time waiting for a response from the external system. Increasing the available threads enables the system to prepare or continue with additional objects.

To change the number of threads, set the `taskThreads` property in the `conf/sync.json` file, for example:

```
    "mappings" : [
        {
            "name" : "systemXmlfileAccounts_managedUser",
            "source" : "system/xmlfile/account",
            "target" : "managed/user",
            "taskThreads" : 20
            ...
         }
    ]
}
```

A value of `0` specifies that reconciliation is run on the main reconciliation thread, that is, in a serial manner.

10.10. Correlation Queries

Every time OpenIDM creates an object through synchronization, it creates a link between the source and target objects. OpenIDM then uses the link to determine the object's situation during later synchronization operations.

Initial, bulk synchronization operations can involve correlating many objects that exist both on source and target systems. In this case, OpenIDM uses correlation queries to find target objects that already exist, and that correspond to source objects. For the target objects that match a correlation query, OpenIDM needs only to create a link, rather than a new target object.

Correlation queries run against target resources. The query syntax therefore depends on the target system, and is either specific to the data store underlying the OpenIDM repository, or to OpenICF query capabilities.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

10.10.1. Managed Object as Correlation Query Target

Queries on managed objects in the repository must be defined in the configuration file for the repository, which is either `openidm/conf/repo.orientdb.json`, or `openidm/conf/repo.jdbc.json`.

The following example shows a correlation query defined in `openidm/conf/repo.orientdb.json`.

```
"for-userName" : "SELECT * FROM ${unquoted:_resource} WHERE userName = ${uid}"
```

By default, a `${value}` token replacement is assumed to be a quoted string. If the value is not a quoted string, use the `unquoted:` prefix, as shown above.

The following correlation query example shows the JavaScript to call the query defined for OrientDB. The `_queryId` property value matches the name of the query specified in `openidm/conf/repo.orientdb.json`, `for-userName`. The `source.name` value replaces `${uid}` in the query. OpenIDM replaces `${unquoted:_resource}` in the query with the name of the table that holds managed objects.

```
{
  "correlationQuery": {
    "type": "text/javascript",
    "source":
      "var query = {'_queryId' : 'for-userName', 'uid' : source.name}; query;"
  }
}
```

The query can return zero or more objects, so the situation OpenIDM assigns to the source object depends on the number of target objects returned.

With a JDBC-based repository, the query defined in `openidm/conf/repo.jdbc.json` is more complex due to how the tables are indexed. The correlation query you define in `openidm/conf/sync.json` is the same, however.

10.10.2. System Object as Correlation Query Target

Correlation queries on system objects access the connector. The connector then executes the query on the external resource.

Your correlation query JavaScript must return a map holding a generic query with the following elements.

- A condition such as "Equals"
- The naming attribute to compare on the system object. In the example that follows, the naming attribute is `uid`.
- The value from the source object to use in the search filter. You set this as the value of the `value` property, which takes an array. In the example that follows, the value to use in the search filter is `source.userName`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
varmap={"query": {"Equals": {"field": "uid", "values": [ source.userName ]}}};
map;
```

10.11. Advanced Data Flow Configuration

Section 10.5, "Basic Data Flow Configuration" shows how to trigger scripts when objects are created and updated. Other situations require you to trigger scripts in response to other synchronization actions. For example, you might not want OpenIDM to delete a managed user directly when an external account is deleted, but instead unlink the objects and deactivate the user in another resource. (Alternatively, you might delete the object in OpenIDM but nevertheless execute a script.) The following example shows a more advanced mapping configuration.

```
 1
 2 {
 3     "mappings": [
 4         {
 5             "name": "systemLdapAccount_managedUser",
 6             "source": "system/ldap/account",
 7             "target": "managed/user",
 8             "validSource": {
 9                 "type": "text/javascript",
10                 "file": "script/isValid.js"
11             },
12             "correlationQuery": {
13                 "type": "text/javascript",
14                 "file": "script/ldapCorrelationQuery.js"
15             },
16             "properties": [
17                 {
18                     "source": "uid",
19                     "transform": {
20                         "type": "text/javascript",
21                         "source": "source.toLowerCase()"
22                     },
23                     "target": "userName"
24                 },
25                 {
26                     "source": "",
27                     "transform": {
28                         "type": "text/javascript",
29                         "source": "if (source.myGivenName)
30                             {source.myGivenName;} else {source.givenName;}"
31                     },
32                     "target": "givenName"
33                 },
34                 {
35                     "source": "",
36                     "transform": {
37                         "type": "text/javascript",
38                         "source": "if (source.mySn)
39                             {source.mySn;} else {source.sn;}"
40                     },
41                     "target": "familyName"
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
 42                 },
 43                 {
 44                     "source": "cn",
 45                     "target": "fullname"
 46                 },
 47                 {
 48                     "comment": "Multi-valued in LDAP, single-valued in AD.
 49                         Retrieve first non-empty value.",
 50                     "source": "title",
 51                     "transform": {
 52                         "type": "text/javascript",
 53                         "file": "script/getFirstNonEmpty.js"
 54                     },
 55                     "target": "title"
 56                 },
 57                 {
 58                     "condition": {
 59                         "type": "text/javascript",
 60                         "source": "var clearObj = openidm.decrypt(object);
 61                             ((clearObj.password != null) &&
 62                             (clearObj.ldapPassword != clearObj.password))"
 63                     },
 64                     "transform": {
 65                         "type": "text/javascript",
 66                         "source": "source.password"
 67                     },
 68                     "target": "__PASSWORD__"
 69                 }
 70             ],
 71             "onCreate": {
 72                 "type": "text/javascript",
 73                 "source": "target.ldapPassword = null;
 74                     target.adPassword = null;
 75                     target.password = null;
 76                     target.ldapStatus = 'New Account'"
 77             },
 78             "onUpdate": {
 79                 "type": "text/javascript",
 80                 "source": "target.ldapStatus = 'OLD'"
 81             },
 82             "onUnlink": {
 83                 "type": "text/javascript",
 84                 "file": "script/triggerAdDisable.js"
 85             },
 86             "policies": [
 87                 {
 88                     "situation": "CONFIRMED",
 89                     "action": "UPDATE"
 90                 },
 91                 {
 92                     "situation": "FOUND",
 93                     "action": "UPDATE"
 94                 },
 95                 {
 96                     "situation": "ABSENT",
 97                     "action": "CREATE"
 98                 },
 99                 {
100                     "situation": "AMBIGUOUS",
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
101                         "action": "EXCEPTION"
102                     },
103                     {
104                         "situation": "MISSING",
105                         "action": "EXCEPTION"
106                     },
107                     {
108                         "situation": "UNQUALIFIED",
109                         "action": "UNLINK"
110                     },
111                     {
112                         "situation": "UNASSIGNED",
113                         "action": "EXCEPTION"
114                     }
115                 ]
116             }
117         ]
118 }
```

The following list shows all the properties that you can use as hooks in mapping configurations to call scripts.

Triggered by Situation

onCreate, onUpdate, onDelete, onLink, onUnlink

Object Filter

vaildSource, validTarget

Correlating Objects

correlationQuery

Triggered on Reconciliation

result

Scripts Inside Properties

condition, transform

Your scripts can get data from any connected system at any time by using the `openidm.read(id)` function, where `id` is the identifier of the object to read.

The following example reads a managed user object from the repository.

```
repoUser = openidm.read("managed/user/ddoe);
```

The following example reads an account from an external LDAP resource.

```
externalAccount = openidm.read("system/ldap/account/uid=ddoe,ou=People,dc=example,dc=com");
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Note that the query targets a DN rather than a UID, as it did in the previous example. The attribute that is used for the `_id` is defined in the connector configuration file and, in this example, is set to `"uidAttribute" : "dn"`. Although it is possible to use a DN (or any unique attribute) for the `_id`, as a best practice, you should use an attribute that is both unique and immutable.

10.12. Scheduling Synchronization

You can schedule synchronization operations, such as LiveSync and reconciliation, using **cron**-like syntax.

This section describes scheduling for reconciliation and LiveSync, however, you can also use OpenIDM's scheduler service to schedule any other event by supplying a link to a script file, in which that event is defined. For information about scheduling other events, and for a deeper understanding of the OpenIDM scheduler service, see *Scheduling Tasks and Events*.

10.12.1. Configuring Scheduled Synchronization

Each scheduled reconciliation and LiveSync task requires a schedule configuration file in `opendidm/conf`. By convention, files are named `openidm/conf/schedule-schedule-name .json`, where *schedule-name* is a logical name for the scheduled synchronization operation, such as `reconcile_systemXmlAccounts_managedUser`.

Schedule configuration files have the following format:

```
{
 "enabled"       : true,
 "persisted"     : false,
 "type"          : "cron",
 "startTime"     : "(optional) time",
 "endTime"       : "(optional) time",
 "schedule"      : "cron expression",
 "misfirePolicy" : "optional, string",
 "timeZone"      : "(optional) time zone",
 "invokeService" : "service identifier",
 "invokeContext" : "service specific context info"
}
```

For an explanation of each of these properties, see *Scheduling Tasks and Events*.

To schedule a reconciliation or LiveSync task, set the `invokeService` property to either `"sync"` (for reconciliation) or `"provisioner"` for LiveSync.

The value of the `invokeContext` property depends on the type of scheduled event. For reconciliation, the properties are set as follows:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
    "invokeService": "sync",
    "invokeContext": {
        "action": "reconcile",
        "mapping": "systemLdapAccount_managedUser"
    }
}
```

The `"mapping"` is either referenced by its name in the `openidm/conf/sync.json` file, or defined inline by using the `"mapping"` property, as shown in the example in *Alternative Mappings*.

For LiveSync, the properties are set as follows:

```
{
    "invokeService": "provisioner",
    "invokeContext": {
        "action": "liveSync",
        "source": "system/OpenDJ/__ACCOUNT__"
    }
}
```

The `"source"` property follows OpenIDM's convention for a pointer to an external resource object and takes the form `system/resource-name/ object-type`.

10.12.2. Alternative Mappings

Mappings for synchronization are usually stored in `openidm/conf/sync.json` for reconciliation, LiveSync, and for pushing changes made to managed objects to external resources. You can, however, provide alternative mappings for scheduled reconciliation by adding the mapping to the schedule configuration instead of referencing a mapping in `sync.json`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
    "enabled": true,
    "type": "cron",
    "schedule": "0 08 16 * * ?",
    "invokeService": "sync",
    "invokeContext": {
        "action": "reconcile",
        "mapping": {
            "name": "CSV_XML",
            "source": "system/Ldap/account",
            "target": "managed/user",
            "properties": [
                {
                    "source": "firstname",
                    "target": "firstname"
                },
                ...
            ],
            "policies": [...]
        }
    }
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 11
Scheduling Tasks and Events

OpenIDM enables you to schedule reconciliation and synchronization tasks. You can also use scheduling to trigger scripts, collect and run reports, trigger workflows, perform custom logging, and so forth.

OpenIDM supports **cron**-like syntax to schedule events and tasks, based on expressions supported by the Quartz Scheduler (bundled with OpenIDM).

If you use configuration files to schedule tasks and events, you must place the schedule files in the `openidm/conf` directory. By convention, OpenIDM uses file names of the form `schedule-schedule-name.json`, where *schedule-name* is a logical name for the scheduled operation, for example, `schedule-reconcile_systemXmlAccounts_managedUser.json`. There are several example schedule configuration files in the `openidm/samples/schedules` directory.

You can configure OpenIDM to pick up changes to scheduled tasks and events dynamically, during initialization and also at runtime. For more information, see *Changing the Configuration*.

In addition to the fine-grained scheduling facility, you can perform a scheduled batch scan for a specified date in OpenIDM data, and then automatically execute a task when this date is reached. For more information, see Section 11.6, "Scanning Data to Trigger Tasks".

11.1. Scheduler Configuration

Schedules are configured through JSON objects. The schedule configuration involves two types of files:

- The `openidm/conf/scheduler.json` file, that configures the overall scheduler service
- One `openidm/conf/schedule-schedule-name.json` file for each configured schedule

The scheduler service configuration file (`openidm/conf/scheduler.json`) has the following format:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
  "threadPool" : {
      "threadCount" : "10"
  },
  "scheduler" : {
      "instanceId" : "scheduler-example",
      "executePersistentSchedules" : "true"
  },
  "advancedProperties" : {
      "org.quartz.scheduler.instanceName" : "OpenIDMScheduler"
  }
}
```

The properties in the `scheduler.json` file relate to the configuration of the Quartz Scheduler.

- `threadCount` specifies the maximum number of threads that are available for the concurrent execution of scheduled tasks.
- `instanceID` can be any string, but must be unique for all schedulers working as if they are the same 'logical' Scheduler within a cluster.
- `executePersistentSchedules` allows you to disable persistent schedule execution for a specific node. If this parameter is set to `false`, the Scheduler Service will support the management of persistent schedules (CRUD operations) but it will not execute any persistent schedules. The value of this property can be a string or boolean and is `true` by default.
- `advancedProperties` (optional) enables you to configure additional properties for the Quartz Scheduler.

For details of all the configurable properties for the Quartz Scheduler, see the *Quartz Scheduler Configuration Reference*.

Each schedule configuration file, `openidm/conf/schedule-` *`schedule-name`*`.json` has the following format:

```
{
 "enabled"              : true,
 "persisted"            : false,
 "concurrentExecution" : false,
 "type"                 : "cron",
 "startTime"            : "(optional) time",
 "endTime"              : "(optional) time",
 "schedule"             : "cron expression",
 "misfirePolicy"        : "optional, string",
 "timeZone"             : "(optional) time zone",
 "invokeService"        : "service identifier",
 "invokeContext"        : "service specific context info",
 "invokeLogLevel"       : "(optional) debug"
}
```

The schedule configuration properties are defined as follows:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

enabled

Set to `true` to enable the schedule. When this property is set to `false`, OpenIDM considers the schedule configuration dormant, and does not allow it to be triggered or executed.

If you want to retain a schedule configuration, but do not want it used, set `enabled` to `false` for task and event schedulers, instead of changing the configuration or **cron** expressions.

persisted (optional)

Specifies whether the schedule state should be persisted or stored in RAM. Boolean (`true` or `false`), `false` by default. For more information, see Section 11.2, "Configuring Persistent Schedules".

concurrentExecution

Specifies whether multiple instances of the same schedule can run concurrently. Boolean (`true` or `false`), `false` by default. Multiple instances of the same schedule cannot run concurrently by default. This setting prevents a new scheduled task from being launched before the same previously launched task has completed. For example, under normal circumstances you would want a liveSync operation to complete its execution before the same operation was launched again. To enable concurrent execution of multiple schedules, set this parameter to `true`. The behavior of "missed" scheduled tasks is governed by the misfirePolicy.

type

Currently OpenIDM supports only `cron`.

startTime (optional)

Used to start the schedule at some time in the future. If this parameter is omitted, empty, or set to a time in the past, the task or event is scheduled to start immediately.

Use ISO 8601 format to specify times and dates (`YYYY-MM-DD Thh:mm :ss`).

endTime (optional)

Used to plan the end of scheduling.

schedule

Takes **cron** expression syntax. For more information, see the *CronTrigger Tutorial* and *Lesson 6: CronTrigger*.

misfirePolicy

For persistent schedules, this optional parameter specifies the behavior if the scheduled task is missed, for some reason. Possible values are as follows:

- `fireAndProceed`. The first execution of a missed schedule is immediately executed when the server is back online. Subsequent executions are discarded. After this, the normal schedule is resumed.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

- `doNothing`, all missed schedules are discarded and the normal schedule is resumed when the server is back online.

timeZone (optional)

If not set, OpenIDM uses the system time zone.

invokeService

Defines the type of scheduled event or action. The value of this parameter can be one of the following:

- `"sync"` for reconciliation
- `"provisioner"` for LiveSync
- `"script"` to call some other scheduled operation defined in a script

invokeContext

Specifies contextual information, depending on the type of scheduled event (the value of the `invokeService` parameter).

The following example invokes reconciliation.

```
{
    "invokeService": "sync",
    "invokeContext": {
        "action": "reconcile",
        "mapping": "systemLdapAccount_managedUser"
    }
}
```

For a scheduled reconciliation task, you can define the mapping in one of two ways:

- Reference a mapping by its name in `sync.json`, as shown in the previous example. The mapping must exist in the `openidm/conf/sync.json` file.
- Add the mapping definition inline by using the `"mapping"` property, as shown in the example in *Alternative Mappings*.

The following example invokes a LiveSync action.

```
{
    "invokeService": "provisioner",
    "invokeContext": {
        "action": "liveSync",
        "source": "system/OpenDJ/__ACCOUNT__"
    }
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

For scheduled LiveSync tasks, the `"source"` property follows OpenIDM's convention for a pointer to an external resource object and takes the form `system/resource-name /object-type`.

The following example invokes a script, which prints the string `Hello World` to the OpenIDM log (`/openidm/logs/openidm0.log.X`) each minute.

```
{
    "invokeService": "script",
    "invokeContext": {
        "script": {
            "type": "text/javascript",
            "source": "java.lang.System.out.println('Hello World');"
        }
    }
}
```

Note that these are sample configurations only. Your own schedule configuration will differ according to your specific requirements.

invokelogLevel (optional)

Specifies the level at which the invocation will be logged. Particularly for schedules that run very frequently, such as LiveSync, the scheduled task can generate significant output to the log file, and the log level should be adjusted accordingly. The default schedule log level is `info`. The value can be set to any one of the SLF4J log levels:

- `"trace"`
- `"debug"`
- `"info"`
- `"warn"`
- `"error"`
- `"fatal"`

11.2. Configuring Persistent Schedules

By default, scheduling information, such as schedule state and details of the schedule execution, is stored in RAM. This means that such information is lost when OpenIDM is rebooted. The schedule configuration itself (defined in the `openidm/conf/schedule- schedule-name.json` file) is not lost when OpenIDM is shut down, and normal scheduling continues when the server is restarted. However, there are no details of missed schedule executions that should have occurred during the period the server was unavailable.

You can configure schedules to be persistent, which means that the scheduling information is stored in the internal repository rather than in RAM. With persistent schedules, scheduling information

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

is retained when OpenIDM is shut down. Any previously scheduled jobs can be rescheduled automatically when OpenIDM is restarted.

Persistent schedules also enable you to manage scheduling across a cluster (multiple OpenIDM instances). When scheduling is persistent, a particular schedule will be executed only once across the cluster, rather than once on every OpenIDM instance. For example, if your deployment includes a cluster of OpenIDM nodes for high availability, you can use persistent scheduling to start a reconciliation action on only one node in the cluster, instead of starting several competing reconciliation actions on each node.

You can use persistent schedules with the default OrientDB repository, or with the MySQL repository (see *Installing a Repository For Production* in the *Installation Guide*).

To configure persistent schedules, set the `"persisted"` property to `true` in the schedule configuration file (`schedule-schedule-name.json)`.

If OpenIDM is down when a scheduled task was set to occur, one or more executions of that schedule might be missed. To specify what action should be taken if schedules are missed, set the `misfirePolicy` in the schedule configuration file. The `misfirePolicy` determines what OpenIDM should do if scheduled tasks are missed. Possible values are as follows:

- `fireAndProceed`. The first execution of a missed schedule is immediately executed when the server is back online. Subsequent executions are discarded. After this, the normal schedule is resumed.
- `doNothing`. All missed schedules are discarded and the normal schedule is resumed when the server is back online.

11.3. Schedule Examples

The following example shows a schedule for reconciliation that is not enabled. When enabled (`"enabled" : true,`), reconciliation runs every 30 minutes, starting on the hour.

```
{
    "enabled": false,
    "persisted": false,
    "type": "cron",
    "schedule": "0 0/30 * * * ?",
    "invokeService": "sync",
    "invokeContext": {
        "action": "reconcile",
        "mapping": "systemLdapAccounts_managedUser"
    }
}
```

The following example shows a schedule for LiveSync enabled to run every 15 seconds, starting at the beginning of the minute. The schedule is persisted, that is, stored in the internal repository rather than in memory. If one or more LiveSync executions are missed, as a result of OpenIDM being unavailable, the first execution of the LiveSync action is executed when the server is back online. Subsequent executions are discarded. After this, the normal schedule is resumed.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
    "enabled": false,
    "persisted": true,
    "misfirePolicy" : "fireAndProceed",
    "type": "cron",
    "schedule": "0/15 * * * * ?",
    "invokeService": "provisioner",
    "invokeContext": {
        "action": "liveSync",
        "source": "system/ldap/account"
    }
}
```

11.4. Checking For Quartz Updates

The Quartz Scheduler can check for updates over the Internet on the Quartz project website, and report available updates in the OpenIDM log. The option is set in `openidm/conf/system.properties` By default, this option is turned off, and should remain off in production.

```
system.properties:org.quartz.scheduler.skipUpdateCheck = true
```

11.5. Service Implementer Notes

Services that can be scheduled implement `ScheduledService`. The service PID is used as a basis for the service identifier in schedule definitions.

11.6. Scanning Data to Trigger Tasks

In addition to the fine-grained scheduling facility described previously, OpenIDM provides a task scanning mechanism. The task scanner enables you to perform a batch scan for a specified date in OpenIDM data, on a scheduled interval, and then to execute a task when this date is reached. When the task scanner identifies a condition that should trigger the task, it can invoke a script created specifically to handle the task.

For example, the task scanner can scan all `managed/user` objects for a "sunset date" and can invoke a script that executes a sunset task on the user object when this date is reached.

11.6.1. Configuring the Task Scanner

The task scanner is essentially a scheduled task that queries a span of managed users. The task scanner is configured in the same way as a regular scheduled task, in a schedule configuration file named (`schedule-*task-name*.json`), with the `"invokeService"` parameter set to `"taskscanner"`. The `"invokeContext"` parameter defines the details of the scan, and the task that should be executed when the specified condition is triggered.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The following example defines a scheduled scanning task that triggers a sunset script. This sample configuration file is provided in the OpenIDM delivery as `openidm/samples/taskscanner/conf/schedule-taskscan_sunset.json`. To use this sample file, you must copy it to the `openidm/conf` directory.

```
{
    "enabled" : true,
    "type" : "cron",
    "schedule" : "0 0 * * * ?",
    "invokeService" : "taskscanner",
    "invokeContext" : {
        "waitForCompletion" : false,
        "maxRecords" : 2000,
        "numberOfThreads" : 5,
        "scan" : {
            "object" : "managed/user",
            "_queryId" : "scan-tasks",
            "property" : "sunset/date",
            "condition" : {
                "before" : "${Time.now}"
            },
            "taskState" : {
                "started" : "sunset/task-started",
                "completed" : "sunset/task-completed"
            },
            "recovery" : {
                "timeout" : "10m"
            }
        },
        "task" : {
            "script" : {
                "type" : "text/javascript",
                "file" : "script/sunset.js"
            }
        }
    }
}
```

The `"invokeContext"` parameter takes the following properties:

`"waitForCompletion"` (optional)

This property specifies whether the task should be performed synchronously. Tasks are performed asynchronously by default (with `waitForCompletion` set to false). A task ID (such as `{"_id":"354ec41f-c781-4b61-85ac-93c28c180e46"}`) is returned immediately. If this property is set to true, tasks are performed synchronously and the ID is not returned until all tasks have completed.

`"maxRecords"` (optional)

The maximum number of records that can be processed. This property is not set by default so the number of records is unlimited. If a maximum number of records is specified, that number will be spread evenly over the number of threads.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`"numberOfThreads"` **(optional)**

By default, the task scanner runs in a multi-threaded manner, that is, numerous threads are dedicated to the same scanning task run. Multithreading generally improves the performance of the task scanner. The default number of threads for a single scanning task is ten. To change this default, set the `"numberOfThreads"` property.

`"scan"`

Defines the details of the scan. The following properties are defined:

`"object"`

Defines the object type against which the query should be performed.

`"_queryId"`

Specifies the query that is performed. The queries that can be set here are defined in the database configuration file (either `conf/repo.orientdb.json` or `conf/repo.jdbc.json`).

`"property"`

Defines the object property against which the range query is performed.

`"condition"` **(optional)**

Indicates the conditions that must be matched for the defined property.

In the previous example, the scanner scans for users for whom the property `sunset/date` is set to a value prior to the current timestamp at the time the script is executed.

You can use these fields to define any condition. For example, if you wanted to limit the scanned objects to a specified location, say, London, you could formulate a query to compare against object locations and then set the condition to be:

```
"condition" : {
    "location" : "London"
},
```

For time-based conditions, the `"condition"` property supports macro syntax, based on the `Time.now` object (which fetches the current time). You can specify any date/time in relation to the current time, using the `+` or `-` operator, and a duration modifier. For example: `"before": "${Time.now + 1d}"` would return all user objects whose `sunset/date` is before tomorrow (current time plus one day). You must include space characters around the operator (`+` or `-`). The duration modifier supports the following unit specifiers:

`s` - second
`m` - minute
`h` - hour
`d` - day

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`M` - month
`y` - year

`"taskState"`

Indicates the fields that are used to track the status of the task.

`"started"` specifies the field that stores the timestamp for when the task begins.
`"completed"` specifies the field that stores the timestamp for when the task completes its operation.

`"recovery"` **(optional)**

Specifies a configurable timeout, after which the task scanner process ends. In a scenario with clustered OpenIDM instances, there might be more than one task scanner running at a time. A task cannot be executed by two task scanners at the same time. When one task scanner "claims" a task, it indicates that the task has been started. That task is then unavailable to be claimed by another task scanner and remains unavailable until the end of the task is indicated. In the event that the first task scanner does not complete the task by the specified timeout, for whatever reason, a second task scanner can pick up the task.

`"task"`

Provides details of the task that is performed. Usually, the task is invoked by a script, whose details are defined in the `"script"` property:

`"type"` - the type of script. Currently, only JavaScript is supported.
`"file"` - the path to the script file. The script file takes at least two objects (in addition to the default objects that are provided to all OpenIDM scripts): `"input"` which is the individual object that is retrieved from the query (in the example, this is the individual user object) and `"objectID"` which is a string that contains the full identifier of the object. The objectID is useful for performing updates with the script as it allows you to target the object directly, for example: `openidm.update(objectID, input['_rev'], input);`. A sample script file is provided in `openidm/samples/taskscanner/script/sunset.js`. To use this sample file, you must copy it to the `openidm/script` directory. The sample script marks all user objects that match the specified conditions as "inactive". You can use this sample script to trigger a specific workflow, or any other task associated with the sunset process. For more information about using scripts in OpenIDM, see the *Scripting Reference*.

11.6.2. Managing Scanning Tasks Over REST

You can trigger, cancel, and monitor scanning tasks over the REST interface, using the REST endpoint `http://localhost:8080/openidm/taskscanner`.

11.6.2.1. Triggering a Scanning Task

The following REST command executes a task named "taskscan_sunrise". The task itself is defined in a file named `openidm/conf/schedule-taskscan_sunset.json`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/taskscanner?_action=execute&name=schedule/taskscan_sunset"
```

By default, a scanning task ID is returned immediately when the task is initiated. Clients can make subsequent calls to the task scanner service, using this task ID to query its state and to call operations on it.

For example, the scanning task initiated previously would return something similar to the following, as soon as it was initiated:

```
{"_id":"edfaf59c-aad1-442a-adf6-3620b24f8385"}
```

To have the scanning task complete before the ID is returned, set the `waitForCompletion` property to `true` in the task definition file (`schedule-taskscan_sunset.json`). You can also set the property directly over the REST interface when the task is initiated. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/taskscanner?_action=execute&name=schedule/
taskscan_sunset&waitForCompletion=true"
```

11.6.2.2. Canceling a Scanning Task

You can cancel a scanning task by sending a REST call with the `cancel` action, specifying the task ID. For example, the following call cancels the scanning task initiated in the previous section.

```
$curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/taskscanner/edfaf59c-aad1-442a-adf6-3620b24f8385?_action=cancel"
```

The output for a scanning task cancelation request is similar to the following, but on a single line:

```
    {"_id":"edfaf59c-aad1-442a-adf6-3620b24f8385",
     "action":"cancel",
     "status":"SUCCESS"}
```

11.6.2.3. Listing Scanning Tasks

You can display a list of scanning tasks that have completed, and those that are in progress, by running a RESTful GET on `"http://localhost:8080/openidm/taskscanner"`. The following example displays all scanning tasks.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/taskscanner"
```

The output of such a request is similar to the following, with one item for each scanning task. The output appears on a single line, but has been indented here, for legibility.


```
{"tasks": [
    {
      "_id": "edfaf59c-aad1-442a-adf6-3620b24f8385",
      "progress": {
        "state": "COMPLETED",
        "processed": 2400,
        "total": 2400,
        "successes": 2400,
        "failures": 0
      },
      "started": 1352455546149,
      "ended": 1352455546182
    }
  ]
}
```


Each scanning task has the following properties:

`_id`

The ID of the scanning task.

`progress`

The progress of the scanning task, summarised in the following fields:

`state` - the overall state of the task, `INITIALIZED`, `ACTIVE`, `COMPLETED`, `CANCELLED`, or `ERROR`
`processed` - the number of processed records
`total` - the total number of records
`successes` - the number of records processed successfully
`failures` - the number of records not able to be processed

`started`

The time at which the scanning task started, .

`ended`

The time at which the scanning task ended.

The number of processed tasks whose details are retained is governed by the `"openidm.taskscanner.maxcompletedruns"` property in the `conf/boot.properties` file. By default, the last one hundred completed tasks are retained.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 12
Managing Passwords

OpenIDM provides password management features that help you enforce password policies, limit the number of passwords users must remember, and let users reset and change their passwords.

12.1. Enforcing Password Policy

A password policy is a set of rules defining what sequence of characters constitutes an acceptable password. Acceptable passwords generally are too complex for users or automated programs to generate or guess.

Password policies set requirements for password length, character sets that passwords must contain, dictionary words and other values that passwords must not contain. Password policies also require that users not reuse old passwords, and that users change their passwords on a regular basis.

OpenIDM enforces password policy rules as part of the general policy service. For more information about the policy service, see *Using Policies to Validate Data*. The default password policy applies the following rules to passwords as they are created and updated:

- A password property is required for any user object.
- The value of a password cannot be empty.
- The password must include at least one capital letter.
- The password must include at least one number.
- The minimum length of a password is 8 characters.
- The password cannot contain the user name, given name, or family name.

You can remove these validation requirements, or include additional requirements, by configuring the policy for passwords. For more information, see *Configuring the Default Policy*.

The password validation mechanism can apply in many situations.

Password change and password reset

Password change involves changing a user or account password in accordance with password policy. Password reset involves setting a new user or account password on behalf of a user.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

By default, OpenIDM controls password values as they are provisioned.

To change the default administrative user password, `openidm-admin`, see the procedure, *To Replace the Default User and Password*, for instructions.

Password recovery

Password recovery involves recovering a password or setting a new password when the password has been forgotten.

OpenIDM provides a self-service end user interface for password changes, password recovery, and password reset. For more information, see *Managing Passwords*.

Password comparisons with dictionary words

You can add dictionary lookups to prevent use of password values that match dictionary words.

Password history

You can add checks to prevent reuse of previous password values

Password expiration

You can configure OpenIDM to call a workflow that ensures users are able to change expiring or to reset expired passwords.

12.2. Password Synchronization

Password synchronization intercepts user password changes, and ensures uniform password changes across resources that store the password. Following password synchronization, the user authenticates using the same password on each resource. No centralized directory or authentication server is required for performing authentication. Password synchronization reduces the number of passwords users need to remember, so they can use fewer, stronger passwords.

OpenIDM can propagate passwords to the resources storing a user's password. OpenIDM can intercept and synchronize passwords changed natively on OpenDJ and Active Directory. See the example in `samples/misc/managed.json` where you installed OpenIDM for a sample password synchronization configuration.

Before using the sample, you must set up OpenDJ and Active Directory, and adjust the password attributes, set in the sample as `ldapPassword` for OpenDJ, `adPassword` for Active Directory, and `password` for the internal OpenIDM password. Also, either set up password policy enforcement on OpenDJ or Active Directory rather than OpenIDM, or ensure that all password policies enforced are identical to prevent password updates on one resource from being rejected by OpenIDM or by another resource.

Also set up password synchronization plugins for OpenDJ and for Active Directory. The password synchronization plugins help by intercepting password changes on the resource before the passwords

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

are stored in encrypted form. The plugins then send intercepted password values to OpenIDM over an encrypted channel.

Procedure 12.1. To Install the OpenDJ Password Synchronization Plugin

Before you start, make sure you configure OpenDJ to communicate over LDAPS as described in the OpenDJ documentation.

The following steps install the plugin in OpenDJ directory server running on the same host as OpenIDM. If you run OpenDJ on a different host use the fully qualified domain name rather than `localhost`, and use your certificates rather than the example.

1. Import the self-signed OpenIDM certificate into the trust store for OpenDJ.

```
$ cd /path/to/OpenDJ/config
$ keytool
 -import
 -alias openidm-localhost
 -keystore truststore
 -storepass `cat keystore.pin`
 -file /path/to/openidm/samples/security/openidm-localhost-cert.txt
Owner: CN=localhost, O=OpenIDM Self-Signed Certificate
Issuer: CN=localhost, O=OpenIDM Self-Signed Certificate
Serial number: 4e4bc38e
Valid from: Wed Aug 17 15:35:10 CEST 2011 until: Tue Aug 17 15:35:10 CEST 2021
Certificate fingerprints:
  MD5:  B8:B3:B4:4C:F3:22:89:19:C6:55:98:C5:DF:47:DF:06
  SHA1: DB:BB:F1:14:55:A0:53:80:9D:62:E7:39:FB:83:15:DA:60:63:79:D1
  Signature algorithm name: SHA1withRSA
  Version: 3
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

2. Download the OpenDJ password synchronization plugin, OPENIDM AGENTS-OPENDJ, from the OpenIDM download page under the ForgeRock Open Stack download page.

3. Unzip the module delivery.

```
$ unzip ~/Downloads/opendj-accountchange-handler-1.0.0.zip
   creating: opendj/
   creating: opendj/config/
   creating: opendj/config/schema/
...
```

4. Copy files to the directory where OpenDJ is installed.

```
$ cd opendj
$ cp -r * /path/to/OpenDJ/
```

5. Restart OpenDJ to load the additional schema from the module.

```
$ cd /path/to/OpenDJ/bin
$ ./stop-ds --restart
```

6. Add the configuration provided to OpenDJ's configuration.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ ./ldapmodify
 --port 1389
 --hostname `hostname`
 --bindDN "cn=Directory Manager"
 --bindPassword "password"
 --defaultAdd
 --filename ../config/openidm-pwsync-plugin-config.ldif
Processing ADD request for cn=OpenIDM Notification Handler,
 cn=Account Status Notification Handlers,cn=config
ADD operation successful for DN cn=OpenIDM Notification Handler
 ,cn=Account Status Notification Handlers,cn=config
```

7. Restart OpenDJ.

```
$ ./stop-ds --restart
...
[16/Jan/2012:15:55:47 +0100] category=EXTENSIONS severity=INFORMATION
 msgID=1049147 msg=Loaded extension from file '/path/to/OpenDJ/lib/extensions
 /opendj-accountchange-handler-1.0.0.jar' (build <unknown>,
 revision <unknown>)
...
[16/Jan/2012:15:55:51 +0100] category=CORE severity=NOTICE msgID=458891 msg=The
 Directory Server has sent an alert notification generated by class
 org.opends.server.core.DirectoryServer (alert type
 org.opends.server.DirectoryServerStarted, alert ID 458887):
 The Directory Server has started successfully
```

8. Enable the plugin for the appropriate password policy.

   The following command enables the plugin for the default password policy.

```
$ ./dsconfig
 set-password-policy-prop
 --port 4444
 --hostname `hostname`
 --bindDN "cn=Directory Manager"
 --bindPassword password
 --policy-name "Default Password Policy"
 --set account-status-notification-handler:"OpenIDM Notification Handler"
 --trustStorePath ../config/admin-truststore
 --no-prompt
```

Procedure 12.2. To Install the Active Directory Password Synchronization Plugin

Use the Active Directory password synchronization plugin to synchronize passwords between OpenIDM and Active Directory (on systems running at least Microsoft Windows 2008).

Install the plugin on Active Directory primary domain controllers (PDCs) to intercept password changes, and send the password values to OpenIDM over an encrypted channel. You must have Administrator privileges to install the plugin. In a clustered Active Directory environment, you must also install the plugin on all PDCs.

1. Download the Active Directory password synchronization plugin, AD CONNECTOR, from the OpenIDM download page under the ForgeRock Open Stack download page.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

2. Unzip the plugin, and double-click `setup.exe` to launch the installation wizard.
3. Complete the installation with the help of the following hints.

   CDDL license agreement

   You must accept the agreement to proceed with the installation.

   OpenIDM URL

   URL where OpenIDM is deployed such as `https://openidm.example.com:8444/openidm` for SSL mutual authentication

   Private Key alias

   Alias used for the OpenIDM certificate also stored in the `keystore.jceks` file, such as `openidm-localhost` used for evaluation

   Private Key password

   Password to access the PFX keystore file, such as `changeit` for evaluation. PFX files contain encrypted private keys, certificates used for authentication and encryption.

   Directory poll interval (seconds)

   Number of seconds between calls to check that Active Directory is available, such as 60

   Query ID parameter

   Query identifier configured in OpenIDM the `openidm/conf/repo.*.json` file. Use `for-userName` for evaluation.

   OpenIDM user password attribute

   Password attribute for the `managed/user` object to which OpenIDM applies password changes

   OpenIDM user search attribute

   The `sAMAccountName` value holder attribute name in the query definition. For example, `"SELECT * FROM ${unquoted:_resource} WHERE userName = ${uid}"`. Use `uid` for the evaluation.

   Select Certificate File

   The PKCS 12 format PFX file containing the certificate used to encrypt communications with OpenIDM. Use `openidm/samples/security/openidm-localhost.p12` for evaluation.

   Select Output Directory

   Select a secure directory where the password changes are queued. The queue contains the encrypted passwords. Yet, the server has to prevent access to this folder except access by the `Password Sync service`. The path name cannot include spaces.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Select Log Directory

The plugin stores logs in the location you select. The path name cannot include spaces.

Select Destination Location

Setup installs the plugin in the location you select, by default `C:\Program Files\OpenIDM Password Sync`.

4. After running the installation wizard, restart the computer.

5. If you must change any settings after installation, access settings using the Registry Editor under HKEY_LOCAL_MACHINE > SOFTWARE > ForgeRock > OpenIDM > PasswordSync.

Procedure 12.3. To Set Up OpenIDM to Handle Password Changes

Follow these steps to configure OpenIDM to access password changes from the directory server.

1. Add the directory server certificate to the OpenIDM trust store so that OpenIDM knows to trust the directory server during mutual authentication.

   The following commands show how to do this with the default OpenDJ and OpenIDM settings.

```
$ cd /path/to/OpenDJ/config/
$ keytool
 -keystore keystore
 -storepass `cat keystore.pin`
 -export
 -alias server-cert
 > /tmp/opendj.crt
$ cd /path/to/openidm/security/
$ keytool
 -import
 -alias opendj-server-cert
 -file /tmp/opendj.crt
 -keystore truststore
 -storepass changeit
 -trustcacerts
Owner: CN=localhost.localdomain, O=OpenDJ Self-Signed Certificate
Issuer: CN=localhost.localdomain, O=OpenDJ Self-Signed Certificate
Serial number: 4f143976
Valid from: Mon Jan 16 15:51:34 CET 2012 until: Wed Jan 15 15:51:34 CET 2014
Certificate fingerprints:
   MD5:  7B:7A:75:FC:5A:F0:65:E5:84:43:6D:10:B9:EA:CC:F0
   SHA1: D1:C6:C9:8A:EA:09:FD:1E:48:BB:B2:F5:95:41:50:2C:AB:4D:0F:C9
   Signature algorithm name: SHA1withRSA
   Version: 3
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

2. Add the configuration to managed objects to handle password synchronization.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

You can find an example for synchronization with both OpenDJ and Active Directory in `samples/misc/managed.json`, JavaScript lines folded for readability:

```
{
    "objects": [
        {
            "name": "user",
            "properties": [
                {
                    "name": "ldapPassword",
                    "encryption": {
                        "key": "openidm-sym-default"
                    }
                },
                {
                    "name": "adPassword",
                    "encryption": {
                        "key": "openidm-sym-default"
                    }
                },
                {
                    "name": "password",
                    "encryption": {
                        "key": "openidm-sym-default"
                    }
                }
            ],
            "onUpdate": {
                "type": "text/javascript",
                "source":
                 "if (newObject.ldapPassword != oldObject.ldapPassword) {
                     newObject.password = newObject.ldapPassword
                  } else if (newObject.adPassword != oldObject.adPassword) {
                      newObject.password = newObject.adPassword
                  }"
            }
        }
    ]
}
```

This sample assumes you define the password as `ldapPassword` for OpenDJ, and `adPassword` for Active Directory.

3. When you change a password in OpenDJ, you will notice that the value changes in OpenIDM.

```
$ tail -f openidm/audit/activity.csv | grep bjensen
...userName=bjensen, ... password={$crypto={...data=tEsy7ZXo6nZtEqzW/uVE/A==...
...userName=bjensen, ... password={$crypto={...data=BReT79lnQEPcvfQG3ibLpg==...
```

Be aware that the plugin is patching the password value of the managed user in OpenIDM. The target `password` property must exist for the patch to work. After the password has been updated in OpenIDM, automatic synchronization is launched and the password is updated in Active Directory.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 13

Managing Authentication, Authorization and RBAC

OpenIDM provides a simple, yet flexible authentication and authorization mechanism based on REST interface URLs and on roles stored in the repository.

13.1. OpenIDM Users

OpenIDM distinguishes between internal users and managed users.

13.1.1. Internal Users

Two internal users are created by default - `anonymous` and `openidm-admin`. These accounts are separated from other user accounts to protect them from any reconciliation or synchronization processes.

OpenIDM stores internal users and their role membership in a table in the repository called `internaluser` when implemented in MySQL, and in the `internal_user` table for an OrientDB repository. You can add or remove internal users over the REST interface (at `http://localhost:8080/openidm/repo/internal/user`) or directly in the repository.

anonymous

This user serves to access OpenIDM anonymously, for users who do not have their own accounts. The anonymous user is primarily intended to allow self-registration.

OpenIDM stores the anonymous user's password, `anonymous`, in clear text in the repository internal user table. The password is not considered to be secret.

openidm-admin

This user serves as the super administrator. After installation, the `openidm-admin` user has full access, and provides a fallback mechanism in case other users are locked out. Do not use `openidm-admin` for normal tasks. Under normal circumstances, no real user is associated with the `openidm-admin` user account, so audit log records that pertain to `openidm-admin` do not reflect the actions of any real person.

OpenIDM encrypts the password, `openidm-admin`, by default. Change the password immediately after installation. For instructions, see *To Replace the Default User and Password*.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

13.1.2. Managed Users

External users that OpenIDM manages are referred to as managed users. When implemented in MySQL, OpenIDM stores managed users in the managed objects table of the repository, named `managedobjects`. A second MySQL table, `managedobjectproperties`, serves as the index table. When implemented in OrientDB, managed objects are stored in the table `managed_user`.

By default, the attribute names for managed user login and password are `userName` and `password`, respectively.

13.2. Authentication

OpenIDM does not allow access to the REST interface unless you authenticate. If a project requires anonymous access, to allow users to self-register for example, then allow access by user `anonymous`, password `anonymous`, as described in Section 13.1.1, "Internal Users". In production, only applications are expected to access the REST interface.

OpenIDM supports an improved authentication mechanism on the REST interface. Unlike basic authentication or form-based authentication, the OpenIDM authentication mechanism is compatible with the AJAX framework.

OpenIDM authentication with standard header fields

```
$ curl --user userName:password
```

This authentication is compatible with standard basic authentication, except that it will not prompt for credentials if they are missing in the request.

OpenIDM authentication with OpenIDM header fields

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
```

For more information about the OpenIDM authentication mechanism, see *Use Message Level Security*.

You can change the attributes that OpenIDM uses to store user login and password values. The attribute names are shown in a database query that is defined in `openidm/conf/repo.repo-type.json`.

Two queries are defined by default.

`credential-internaluser-query`

Uses the `_openidm_id` attribute for login

`credential-query`

Uses the `userName` attribute for login

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The openidm/conf/authentication.json file defines the currently active query as the value of the queryId property. In the following example, credential-query is active.

```
{
    "queryId" : "credential-query",
    "queryOnResource" : "managed/user",
    "defaultUserRoles" : [ ]
}
```

You can explicitly define the properties that constitute passwords or roles by setting the propertyMapping object in the conf/authentication.json file. By default, the property mapping is configured as follows:

```
...
    "propertyMapping" : {
        "userId" : "_id",
        "userCredential" : "password",
        "userRoles" : "roles"
    },
...
```

13.3. Roles

OpenIDM sets up the following roles by default.

openidm-reg

Role for users accessing OpenIDM with the default anonymous account

openidm-admin

OpenIDM administrator role

openidm-authorized

Default role for any user authenticated with a user name and password

openidm-cert

Default role for any user authenticated with mutual SSL authentication

You configure the default roles that are assigned to successfully authenticated users by setting the defaultUserRoles property in openidm/conf/authentication.json, which takes a list. The default value is openidm-authorized.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
    "queryId": "credential-query",
    "queryOnResource": "managed/user",
    "defaultUserRoles": [
        "openidm-authorized"
    ]
}
```

13.4. Authorization

OpenIDM provides role-based authorization that restricts direct HTTP access to REST interface URLs. The default authorization configuration grants different access rights to users that are assigned the roles `"openidm-admin"`, `"openidm-cert"`, `"openidm-authorized"`, and `"openidm-reg"`.

Note that this access control applies to direct HTTP calls only. Access for internal calls (for example, calls from scripts) is not affected by this mechanism.

Authorization is configured in two script files:

- `openidm/bin/defaults/script/router-authz.js`
- `openidm/script/access.js`

OpenIDM calls these scripts for each request, via the `onRequest` hook that is defined in the default `router.json` file. The scripts either throw the string `Access denied`, or nothing. If `Access denied` is thrown, OpenIDM denies the request.

13.4.1. `router-authz.js`

This file provides the functions that enforce access rules. For example, the following function controls whether users with a certain role can start a specified process.

```
...
function isAllowedToStartProcess() {
var processDefinitionId = request.value._processDefinitionId;
return isProcessOnUsersList(processDefinitionId);
}
...
```

There are certain functions in `router-authz.js` that should *not* be altered. These are indicated in the file itself.

13.4.2. `access.js`

This file defines the access configuration for HTTP requests and references the methods defined in `router-authz.js`. Each entry in the configuration contains a pattern to match against the incoming

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

request ID, and the associated roles, methods, and actions that are allowed for requests on that pattern.

The following sample configuration entry indicates the configurable parameters and their purpose.

```
{
    "pattern"   : "*",
    "roles"     : "openidm-admin",
    "methods"   : "*", // default to all methods allowed
    "actions"   : "*", // default to all actions allowed
    "customAuthz" : "disallowQueryExpression()",
    "excludePatterns": "system/*"
},
```

The overall intention of this entry is to allow users with the role `openidm-admin` HTTP access to everything except the `system` endpoints. The parameters are as follows:

- `"pattern"` - the REST endpoint to which access is being controlled. `"*"` indicates access to all endpoints. `"managed/user/*"` would indicate access to all managed user objects.
- `"roles"` - a comma-separated list of the roles to which this access configuration applies.
- `"methods"` - a comma separated list of the methods to which access is being granted. The method can be one or more of `create, read, update, delete, patch, action, query`. A value of `"*"` indicates that all methods are allowed. A value of `""` indicates that no methods are allowed.
- `"actions"` - a comma separated list of the allowed actions. The possible values depend on the service (URL) that is being exposed. The following list indicates the possible actions for each service.

  `openidm/managed` - `patch`
  `openidm/recon` - `recon, cancel`
  `openidm/sync` - `onCreate, onUpdate, onDelete, recon, performAction`
  `openidm/external/email` - `(no action parameter applies)`
  `openidm/external/rest` - `(no action parameter applies)`
  `openidm/authentication` - `reauthenticate`
  `openidm/system` - `createconfiguration`
  `openidm/system/*` - `script`
  `openidm/taskscanner` - `execute, cancel`
  `openidm/workflow/processinstance` - `(no action parameter applies)`
  `openidm/workflow/taskinstance` - `claim,complete`

  A value of `"*"` indicates that all actions exposed for that service are allowed. A value of `""` indicates that no actions are allowed.

- `"customAuthz"` - an optional parameter that enables you to specify a custom function for additional authorization checks. These functions are defined in `router-authz.js` .

  The `allowedPropertiesForManagedUser` variable, declared at the beginning of the file, enables you to create a white list of attributes that users may modify on their own accounts.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

- `"excludePatterns"` - an optional parameter that enables you to specify particular endpoints to which access should not be given.

13.4.3. Extending the Authorization Mechanism

You can extend the default authorization mechanism by defining additional functions in `router-authz.js` and by creating new access control configuration definitions in `access.js`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 14

Securing & Hardening OpenIDM

After following the guidance in this chapter, make sure that you test your installation to verify that it behaves as expected before putting it into production.

Out of the box, OpenIDM is set up for ease of development and deployment. When deploying OpenIDM in production, take the following precautions.

14.1. Use SSL and HTTPS

Disable plain HTTP access, included for development convenience, as described in the section titled *Secure Jetty*.

Use TLS/SSL to access OpenIDM, ideally with mutual authentication so that only trusted systems can invoke each other. TLS/SSL protects data on the wire. Mutual authentication with certificates imported into the applications' trust and key stores provides some confidence for trusting application access.

Augment this protection with message level security where appropriate.

14.2. Restrict REST Access to the HTTPS Port

Use certificates to secure REST access, over HTTPS. The following procedure shows how to generate a self-signed certificate to secure REST calls, over HTTPS. Note that in production systems, it is recommended that you use a key that has been signed by a certificate authority.

1. Extract the certificate that is installed with OpenIDM.

   ```
   $ openssl s_client -showcerts -connect localhost:8443 </dev/null
   ```

   This command outputs the entire certificate to the terminal.

2. Using any text editor, create a file named `server.crt`. Copy the portion of the certificate from `BEGIN CERTIFICATE` to `END CERTIFICATE` and paste it into the `server.crt` file. Your `server.crt` file should now contain something like the following:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ more server.crt
-----BEGIN CERTIFICATE-----
MIIB8zCCAVygAwIBAgIETkvDjjANBgkqhkiG9w0BAQUFADA+MSgwJgYDVQQKEx9P
cGVuSURNIFNlbGYtU2lnbmVkIENlcnRpZmljYXRlMRIwEAYDVQQDEwlsb2NhbGhv
c3QwHhcNMTEwODE3MTMzNTEwWhcNMjEwODE3MTMzNTEwWjA+MSgwJgYDVQQKEx9P
cGVuSURNIFNlbGYtU2lnbmVkIENlcnRpZmljYXRlMRIwEAYDVQQDEwlsb2NhbGhv
c3QwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAKwMkyvHS5yHAnI7+tXUIbfI
nQfhcTChpWNPTHc/cli/+Ta1InTpN8vRScPoBG0BjCaIKnVVl2zZ5ya74UKgwAVe
oJQ0xDZvIyeC9PlvGoqsdtH/Ihi+T+zzZ14oVxn74qWoxZcvkG6rWEOd42QzpVhg
wMBzX98slxkOZhG9IdRxAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEASo4qMI0axEKZ
m0jU4yJejLBHydWoZVZ8fKcHVlD/rTirtVgWsVgvdr3yUr0Idk1rH1nEF47Tzn+V
UCq7qJZ75HnIIeVrZqmfTx8169paAKAaNF/KRhTE6ZII8+awst02L86shSSWqWz3
s5xPB2YTaZHWWdzrPVv90gL8JL/N7/Q=
-----END CERTIFICATE-----
```

3. Generate a private, self-signed key as follows:

    a. Generate an encrypted 1024-bit RSA key, and save it to a file named `localhost.key`. Enter a pass phrase for the key as requested.

    ```
    $ openssl genrsa des3 out localhost.key 1024
    Generating RSA private key, 1024 bit long modulus
    .........++++++
    ........................++++++
    e is 65537 (0x10001)
    Enter pass phrase for localhost.key:
    Verifying - Enter pass phrase for localhost.key:
    ```

    b. Generate a certificate request using the key you created in the previous step, and save it to a file named `localhost.csr`. Enter any required information to create the DN for the request.

    ```
    $ openssl req new key localhost.key out localhost.csr
    ```

    This step creates a file, `localhost.csr`, that contains the details of the certificate request.

    c. Sign the certificate with the key you created in the previous step, and generate a certificate that is valid for one year in a file named `localhost.crt`. The `x509` subcommand enables you to retrieve the information that is stored in the SSL certificate. Output will depend on the details that you entered in the certificate request.

    ```
    $ openssl x509 req days 365 in localhost.csr signkey localhost.key out localhost.crt
    Signature ok
    subject=/C=FR/ST=Il-DE-FRANCE/L=Paris/O=example.com
    Getting Private key
    Enter pass phrase for localhost.key:
    ```

    The contents of `localhost.crt` should now be something like this:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ more localhost.crt
-----BEGIN CERTIFICATE-----
MIIB/zCCAWgCCQD6VdiF6rX2czANBgkqhkiG9w0BAQUFADBEMQswCQYDVQQGEwJa
QTELMAkGA1UECBMCV0MxEjAQBgNVBAcTCUNhcGUgVG93bjEUMBIGA1UEChMLZXhh
bXBsZS5jb20wHhcNMTMwMTI1MTIzNzIyWhcNMTQwMTI1MTIzNzIyWjBEMQswCQYD
VQQGEwJaQTELMAkGA1UECBMCV0MxEjAQBgNVBAcTCUNhcGUgVG93bjEUMBIGA1UE
ChMLZXhhbXBsZS5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAONLO82s
wKA0tWkbR66DajwQKNO9QlYwZvcK4X7MFOcwex+8j2vvG5HCB0BW2Gm72mFTWei8
gVgQDP1oe/yTWDZRaiJ8rGWdvpgH1Cmxcd3N1AhhRya1I2j5wxrc9ZsyyTYCg2fd
pFfULrUXSd9QlB2qQZz7kb4ksT/mSwPiGqvFAgMBAAEwDQYJKoZIhvcNAQEFBQAD
gYEA3WrP8NKjXwQzE0vabYmdUhPHt3PF8EMMwVJ+h8G9Dwmtll0P/kLybXdHF1P/
SvN8ofdaEKi4DrLvBifkJvHdTm9DgZJo+bROM6LM9kac6CxNvwj9m/4g6mhnjxEV
63WQPzvAeri051JC0ysMVe5vf+l00t+J8W6SfPTKwoXDQhY=
-----END CERTIFICATE-----
```

4. Combine the contents of `server.crt` and `localhost.crt` to create a Privacy Enhanced Mail Certificate (`.pem`) file named `CA.pem`.

   ```
   $ cat server.crt localhost.crt > CA.pem
   ```

   The contents of `CA.pem` should be something like the following (a concatenation of `server.crt` and `localhost.crt`).

   ```
   $ more CA.pem
   -----BEGIN CERTIFICATE-----
   MIIB8zCCAVygAwIBAgIETkvDjjANBgkqhkiG9w0BAQUFADA+MSgwJgYDVQQKEx9P
   cGVuSURNIFNlbGYtU2lnbmVkIENlcnRpZmljYXRlMRIwEAYDVQQDEwlsb2NhbGhv
   c3QwHhcNMTEwODE3MTMzNTEwWhcNMjEwODE3MTMzNTEwWjA+MSgwJgYDVQQKEx9P
   cGVuSURNIFNlbGYtU2lnbmVkIENlcnRpZmljYXRlMRIwEAYDVQQDEwlsb2NhbGhv
   c3QwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAKwMkyvHS5yHAnI7+tXUIbfI
   nQfhcTChpWNPTHc/cli/+Ta1InTpN8vRScPoBG0BjCaIKnVVl2zZ5ya74UKgwAVe
   oJQ0xDZvIyeC9PlvGoqsdtH/Ihi+T+zzZ14oVxn74qWoxZcvkG6rWEOd42QzpVhg
   wMBzX98slxkOZhG9IdRxAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEASo4qMI0axEKZ
   m0jU4yJejLBHydWoZVZ8fKcHVlD/rTirtVgWsVgvdr3yUr0Idk1rH1nEF47Tzn+V
   UCq7qJZ75HnIIeVrZqmfTx8169paAKAaNF/KRhTE6ZII8+awst02L86shSSWqWz3
   s5xPB2YTaZHWWdzrPVv90gL8JL/N7/Q=
   -----END CERTIFICATE-----
   -----BEGIN CERTIFICATE-----
   MIIB/zCCAWgCCQD6VdiF6rX2czANBgkqhkiG9w0BAQUFADBEMQswCQYDVQQGEwJa
   QTELMAkGA1UECBMCV0MxEjAQBgNVBAcTCUNhcGUgVG93bjEUMBIGA1UEChMLZXhh
   bXBsZS5jb20wHhcNMTMwMTI1MTIzNzIyWhcNMTQwMTI1MTIzNzIyWjBEMQswCQYD
   VQQGEwJaQTELMAkGA1UECBMCV0MxEjAQBgNVBAcTCUNhcGUgVG93bjEUMBIGA1UE
   ChMLZXhhbXBsZS5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAONLO82s
   wKA0tWkbR66DajwQKNO9QlYwZvcK4X7MFOcwex+8j2vvG5HCB0BW2Gm72mFTWei8
   gVgQDP1oe/yTWDZRaiJ8rGWdvpgH1Cmxcd3N1AhhRya1I2j5wxrc9ZsyyTYCg2fd
   pFfULrUXSd9QlB2qQZz7kb4ksT/mSwPiGqvFAgMBAAEwDQYJKoZIhvcNAQEFBQAD
   gYEA3WrP8NKjXwQzE0vabYmdUhPHt3PF8EMMwVJ+h8G9Dwmtll0P/kLybXdHF1P/
   SvN8ofdaEKi4DrLvBifkJvHdTm9DgZJo+bROM6LM9kac6CxNvwj9m/4g6mhnjxEV
   63WQPzvAeri051JC0ysMVe5vf+l00t+J8W6SfPTKwoXDQhY=
   -----END CERTIFICATE-----
   ```

5. Test REST access on the HTTPS port, using the certificate that you created in the previous step. For example:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ curl
 --header "X-OpenIDM-Username:openidm-admin"
 --header "X-OpenIDM-Password:openidm-admin"
 --cacert CA.pem
 --request GET
 "https://localhost:8443/openidm/managed/user/?_queryId=query-all-ids
{
    "conversion-time-ms": 0,
    "result": [
        {
            "_rev": "0",
            "_id": "8afd44a7-13be-449e-9c47-7a310e675c00"
        }
    ],
    "query-time-ms": 1
}
```

Note

If you receive the response `curl: (52) Empty reply from server`, check that you have, in fact, used `https` and not `http` in the URL.

14.3. Encrypt Data Internally & Externally

Beyond relying on end-to-end availability of TLS/SSL to protect data, OpenIDM also supports explicit encryption of data that goes on the wire. This can be important if the TLS/SSL termination happens prior to the final end point.

OpenIDM also supports encryption of data stored in the repository, using a symmetric key. This protects against some attacks on the data store.

OpenIDM automatically encrypts sensitive data in configuration files, such as passwords. OpenIDM replaces clear text values when the system first reads the configuration file. Take care with configuration files having clear text values that OpenIDM has not yet read and updated.

14.4. Use Message Level Security

OpenIDM supports message level security, forcing authentication before granting access. Authentication works by means of a filter-based mechanism that lets you use either an HTTP Basic like mechanism or OpenIDM-specific headers, setting a cookie in the response that you can use for subsequent authentication. If you attempt to access OpenIDM URLs without the appropriate headers or session cookie, OpenIDM returns HTTP 401 Unauthorized. If you use a session cookie, you must include an additional header that indicates the origin of the request.

The following examples show successful authentications.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ curl
 --dump-header /dev/stdout
 --user openidm-admin:openidm-admin
 "http://localhost:8080/openidm/managed/user/?_queryId=query-all-ids"

HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=2l0zobpuk6st1b2m7gvhg5zas;Path=/
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json; charset=UTF-8
Date: Wed, 18 Jan 2012 10:36:19 GMT
Accept-Ranges: bytes
Server: Restlet-Framework/2.0.9
Transfer-Encoding: chunked

{"query-time-ms":1,"result":[{"_id":"ajensen"},{"_id":"bjensen"}]}

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

HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=ixnekr105coj11ji67xcluux8;Path=/
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json; charset=UTF-8
Date: Wed, 18 Jan 2012 10:36:40 GMT
Accept-Ranges: bytes
Server: Restlet-Framework/2.0.9
Transfer-Encoding: chunked

{"query-time-ms":0,"result":[{"_id":"ajensen"},{"_id":"bjensen"}]}

$ curl
 --dump-header /dev/stdout
 --header "Cookie: JSESSIONID=ixnekr105coj11ji67xcluux8"
 --header "X-Requested-With: OpenIDM Plugin"
 "http://localhost:8080/openidm/managed/user/?_queryId=query-all-ids"

HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Date: Wed, 18 Jan 2012 10:37:20 GMT
Accept-Ranges: bytes
Server: Restlet-Framework/2.0.9
Transfer-Encoding: chunked

{"query-time-ms":1,"result":[{"_id":"ajensen"},{"_id":"bjensen"}]}
```

Notice that the last example uses the cookie OpenIDM set in the response to the previous request, and includes the `X-Requested-With` header to indicate the origin of the request. The value of the header can be any string, but should be informative for logging purposes. If you do not include the `X-Requested-With` header, OpenIDM returns HTTP 403 Forbidden.

You can also request one-time authentication without a session.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

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

HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Date: Wed, 18 Jan 2012 10:52:27 GMT
Accept-Ranges: bytes
Server: Restlet-Framework/2.0.9
Transfer-Encoding: chunked

{"query-time-ms":1,"result":[{"_id":"ajensen"},{"_id":"bjensen"}]}
```

To log out and destroy the session, send the specific OpenIDM header.

```
$ curl
 --dump-header /dev/stdout
 --header "Cookie: JSESSIONID=ixnekr105coj11ji67xcluux8"
 --header "X-Requested-With: OpenIDM Plugin"
 --header "X-OpenIDM-Logout: true"
 "http://localhost:8080/openidm/"

HTTP/1.1 204 No Content
```

OpenIDM creates the `openidm-admin` user with password `openidm-admin` by default. This internal user is stored in OpenIDM's repository.

```
mysql> select objectid,roles from internaluser;
+---------------+---------------------------------+
| objectid      | roles                           |
+---------------+---------------------------------+
| anonymous     | openidm-reg                     |
| openidm-admin | openidm-admin,openidm-authorized |
+---------------+---------------------------------+
2 rows in set (0.00 sec)
```

OpenIDM uses the internal table for authentication, and also to set the roles for RBAC authorization of an authenticated user. The router service, described in the *Router Service Reference* appendix, enables you to apply filters as shown in `openidm/conf/router.json` and the associated script, `openidm/script/router-authz.js`. See the chapter on *Managing Authentication, Authorization & RBAC* for details.

14.5. Replace Default Security Settings

The default security settings are adequate for evaluation purposes. For production, change the default encryption key, and then replace the default user password.

Procedure 14.1. To Change Default Encryption Keys

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

By default, OpenIDM uses an symmetric encryption key with alias `openidm-sym-default`. Change this default key before deploying OpenIDM in production.

1. Add the new key to the key store.

```
$ cd /path/to/openidm/
$ keytool
 -genseckey
 -alias new-sym-key
 -keyalg AES
 -keysize 128
 -keystore security/keystore.jceks
 -storetype JCEKS
Enter keystore password:
Enter key password for <new-sym-key>
  (RETURN if same as keystore password):
Re-enter new password:
$
```

   Also see `openidm/samples/security/keystore_readme.txt`.

2. Change the alias used in `openidm/conf/boot/boot.properties`.

Procedure 14.2. To Replace the Default User & Password

After changing the default encryption key, change at least the default user password.

1. Use the **encrypt** command to obtain the encrypted version of the new password.

```
$ cd /path/to/openidm/
$ cli.sh encrypt newpwd
...
-----BEGIN ENCRYPTED VALUE-----
{
  "$crypto" : {
    "value" : {
      "iv" : "TCoC/YrmiRmINw6jCPB5LQ==",
      "data" : "nCFvBIApIQ7C6k+UPzosaA==",
      "cipher" : "AES/CBC/PKCS5Padding",
      "key" : "openidm-sym-default"
    },
    "type" : "x-simple-encryption"
  }
}
------END ENCRYPTED VALUE------
```

2. Replace the user object in the `openidm/db/scripts/mysql/openidm.sql` script before setting up MySQL as a repository for OpenIDM.

   Alternatively, replace the user in the internal user table.

14.6. Secure Jetty

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Before running OpenIDM in production, edit the `openidm/conf/jetty.xml` configuration to avoid clear text HTTP. Opt instead for HTTPS, either with or without mutual authentication. To disable plain HTTP access, comment out the section in `openidm/conf/jetty.xml` that enables HTTP on port 8080.

```
<!--
<Item>
    <New class="org.eclipse.jetty.server.nio.SelectChannelConnector">
        <Set name="host"><Property name="jetty.host" /></Set>
        <Set name="port">8080</Set>
        <Set name="maxIdleTime">300000</Set>
        <Set name="Acceptors">2</Set>
        <Set name="statsOn">false</Set>
        <Set name="confidentialPort">8443</Set>
        <Set name="lowResourcesConnections">20000</Set>
        <Set name="lowResourcesMaxIdleTime">5000</Set>
    </New>
</Item>
-->
```

14.7. Protect Sensitive REST Interface URLs

Although the repository is accessible directly by default, since anything attached to the router is accessible with the default policy, avoid direct HTTP access in production. If you do not need such access, deny it in the authorization policy to reduce the attack surface.

Similarly deny direct HTTP access to system objects in production, particularly access to `action`. As a rule of thumb, do not expose anything that is not used in production. The main public interfaces over HTTP are `/openidm/managed/` and `/openidm/config/`. Other URIs are triggered indirectly, or are for internal consumption.

OpenIDM supports native query expressions on the JDBC repository and it is possible to enable these over HTTP, for example:

```
$curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 "http://localhost:8080/openidm/managed/user?_queryExpression=select+*+from+managedobjects"
```

By default, direct HTTP access to native queries is disallowed, and should remain so in production systems. To enable native queries on the JDBC repository over HTTP, specifically for testing or development purposes, remove the custom authorization call from the router authorization script (`openidm/script/router-authz.js`).

```
"customAuthz" : "disallowQueryExpression()"
```

Remember to remove the comma at the end of the preceding line as well.

See the chapter on *Managing Authentication, Authorization & RBAC* for an example showing how to protect sensitive URLs.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

14.8. Protect Sensitive Files & Directories

Protect OpenIDM files from access by unauthorized users.

In particular, prevent other users from reading files in at least the `openidm/conf/boot/` and `openidm/security/` directories.

14.9. Obfuscate Bootstrap Information

OpenIDM uses the information in `conf/boot/boot.properties` including the key store password to start up. You can set an obfuscated version in the file, or prompt for the password at start up time.

To generate obfuscated versions of a password, use the **encrypt** command-line utility. For more information, see the *encrypt* documentation.

14.10. Remove or Protect Development & Debug Tools

Before deploying OpenIDM in production, remove or protect development and debug tools, including the OSGi console exposed under `/system/console`. Authentication for this console is not integrated with authentication for OpenIDM.

To remove the OSGi console, remove the web console bundle, `org.apache.felix.webconsole-version.jar`.

If you cannot remove the OSGi console, then protect it by overriding the default `admin:admin` credentials. Create a file called `openidm/conf/org.apache.felix.webconsole.internal.servlet.OsgiManager.cfg` containing the user name and password to access the console in Java properties file format.

```
username=user-name
password=password
```

14.11. Protect the OpenIDM Repository

Use the JDBC repository. OrientDB is not yet supported for production use.

Use a strong password for the JDBC connection. Do not rely on default passwords.

Use a case sensitive database, particularly if you work with systems with different identifiers that match except for case. Otherwise correlation queries can pick up identifiers that should not be considered the same.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

14.12. Adjust Log Levels

Leave log levels at `INFO` in production to ensure that you capture enough information to help diagnose issues. See the chapter on *Configuring Server Logs* for more information.

At start up and shut down, `INFO` can produce many messages. Yet, during stable operation, `INFO` generally results in log messages only when coarse-grain operations such as scheduled reconciliation start or stop.

14.13. Set Up Restart At System Boot

You can run OpenIDM in the background as a service (daemon), and add startup and shutdown scripts to manage the service at system boot and shutdown. For more information, see *Starting and Stopping OpenIDM*.

See your operating system documentation for details on adding a service such as OpenIDM to be started at boot and shut down at system shutdown.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 15
Integrating Business Processes and Workflows

Key to any identity management solution is the ability to provide workflow-driven provisioning activities, whether for self-service actions such as requests for entitlements, roles or resources, running sunrise or sunset processes, handling approvals with escalations, or performing maintenance.

OpenIDM provides an embedded workflow and business process engine based on Activiti and the Business Process Model and Notation (BPMN) 2.0 standard.

More information about Activiti and the Activiti project can be found at http://www.activiti.org.

15.1. BPMN 2.0 and the Activiti Tools

Business Process Model and Notation 2.0 is the result of consensus among Business Process Management (BPM) system vendors. The Object Management Group (OMG) has developed and maintained the BPMN standard since 2004.

The first version of the BPMN specification focused only on graphical notation, and quickly became popular with the business analyst audience. BPMN 1.x defines how constructs such as human tasks, executable scripts, and automated decisions are visualized in a vendor-neutral, standard way. The second version of BPMN extends that focus to include execution semantics, and a common exchange format. Thus, BPMN 2.0 process definition models can be exchanged not only between different graphical editors, but can also be executed as is on any BPMN 2.0-compliant engine, such as the engine embedded in OpenIDM.

Using BPMN 2.0, you can add artifacts describing workflow and business process behavior to OpenIDM for provisioning and other purposes. For example, you can craft the actual artifacts defining business processes and workflow in a text editor, or using a special Eclipse plugin. The Eclipse plugin provides visual design capabilities, simplifying packaging and deployment of the artifact to OpenIDM. See the Activiti BPMN 2.0 Eclipse Plugin documentation for instructions on installing Activiti Eclipse BPMN 2.0 Designer.

Also, read the Activiti *User Guide* section covering *BPMN 2.0 Constructs*, which describes in detail the graphical notations and XML representations for events, flows, gateways, tasks, and process constructs.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

15.2. Setting Up Activiti Integration With OpenIDM

There are two modes of integrating Activiti with OpenIDM:

- Local integration, where an embedded Activiti Process Engine is started in the OpenIDM OSGi container.
- Remote integration, where OpenIDM and Activiti run as separate instances and the integration is done using a REST API.

15.2.1. Setting Up Local Integration

The embedded workflow and business process engine is provided as part of the standard OpenIDM build.

Install the OpenIDM build, as described in the *Installation Guide* in the *Installation Guide*. Start OpenIDM, and run the **scr list** command at the console to check that the workflow bundle is active.

```
-> scr list
...
[  14] [active       ] org.forgerock.openidm.workflow
...
```

To verify the workflow integration you need at least one workflow definition in the `/path/to/openidm/workflow` directory. A sample workflow (`example.bpmn20.xml`) is provided in the `/path/to/openidm/samples/misc` directory. Copy this workflow to the `/path/to/openidm/workflow` directory to test the workflow integration.

```
$ cd /path/to/openidm
$ cp samples/misc/example.bpmn20.xml workflow/
```

You can verify the workflow integration by using the REST API. The following REST call lists the defined workflows:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/processdefinition?_queryId=query-all-ids"
```

The sample workflow definition that you copied in the previous step is named `osgiProcess`. The result of the preceding REST call is therefore something like:

```
{"result":[
     {"_id":"osgiProcess:1:3",
      "name":"Osgi process"
     }
        ]
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The `osgiProcess` definition calls OpenIDM, queries the available workflow definitions from Activiti, then prints the list of workflow definitions to the OpenIDM logs. Invoke the `osgiProcess` workflow with the following REST call to OpenIDM:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/workflow/processinstance?_action=createProcessInstance"
 --data '{"_key":"osgiProcess"}'
```

The workflow prints the list of workflow definitions to the OpenIDM console. With the default sample, you should see something like this on the console:

```

script task using resolver: [
  result:[
    [_id:osgiProcess:1:3, name:Osgi process]
  ]
]
script task using expression resolver: [
  result:[
    [_id:osgiProcess:1:3, name:Osgi process]
  ]
]
```

15.2.1.1. Setting Up Remote Integration

You can set up integration with a remote Activiti engine, as described in the following steps.

1. Download and install OpenIDM, as described in the *Installation Guide* in the *Installation Guide*.
2. Download and unzip the Activiti zip file (`activiti-5.10.zip`) from Activiti Downloads page.
3. Edit the Activiti configuration file to avoid a port conflict with OpenIDM. OpenIDM runs on port 8080 by default. Edit the file so that Activiti Explorer runs on port 9090 (by replacing each instance of 8080 with 9090). The following example uses `sed` on a UNIX system to replace all instances of 8080 with 9090.

   ```
   $ cd /path/to/activiti/setup/
   $ sed -i.bak 's/8080/9090/g' build.xml
   ```

   Note

   There is currently a bug in the Activiti demo which might mean that all port replacements are not made. If you cannot access Activiti Explorer (in the next step) after making this change, *also* edit the following file to find and replace each instance of 8080 with 9090: `/path/to/activiti/apps/apache-tomcat-6.0.32/conf/server.xml`.

4. Set up the default Activiti demo.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ cd /path/to/activiti/setup/
$ ant demo.start
```

5. In a browser, check that Activiti Explorer is running (on localhost:9090/activiti-explorer). Log in with the default userid `kermit` and password `kermit`. If all is well, log out.

6. Configure Tomcat to operate with OpenIDM.

   1. Stop Tomcat.

      ```
      $ cd /path/to/activiti/setup/
      $ ant tomcat.stop
      ```

   2. Copy the OpenIDM remote workflow WAR file to the Tomcat webapps folder.

      ```
      $ cd /path/to/openidm/bin/workflow
      $ cp openidm-workflow-remote-2.1.2.war \
       /path/to/activiti/apps/apache-tomcat-6.0.32/webapps/
      ```

   3. Copy the OpenIDM workflow Activiti demo jar file to the Tomcat Activiti Explorer library.

      ```
      $ cd /path/to/openidm/bin/workflow/
      $ cp openidm-workflow-activiti-2.1.2-jar-with-dependencies.jar \
       /path/to/activiti/apps/apache-tomcat-6.0.32/webapps/activiti-explorer/WEB-INF/lib/
      ```

   4. Edit the Activiti Explorer configuration file to be able to use the OpenIDM extensions.

      ```
      $ cd /path/to/activiti/
      $ vi apps/apache-tomcat-6.0.32/webapps/activiti-explorer/WEB-INF/applicationContext.xml
      ```

      Replace the `processEngineConfiguration` with the OpenIDM extended configuration. So remove this section:

      ```
        <bean id="processEngineConfiguration" class="org.activiti.spring.SpringProcessEngineConfiguration">
              <property name="dataSource" ref="dataSource" />
              <property name="transactionManager" ref="transactionManager" />
              <property name="databaseSchemaUpdate" value="true" />
              <property name="jobExecutorActivate" value="true" />
              <property name="customFormTypes">
               <list>
                <ref bean="userFormType"/>
               </list>
              </property>
             </bean>
      ```

      and replace it with this section:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
    <bean id="processEngineConfiguration" class="org.activiti.spring.SpringProcessEngineConfiguration">
     <property name="dataSource" ref="dataSource" />
     <property name="transactionManager" ref="transactionManager" />
     <property name="databaseSchemaUpdate" value="true" />
     <property name="jobExecutorActivate" value="true" />
     <property name="customFormTypes">
       <list>
        <ref bean="userFormType"/>
       </list>
      </property>
    <property name="customSessionFactories">
      <list>
        <bean class="org.forgerock.openidm.workflow.activiti.impl.session.OpenIDMSessionFactory">
          <property name="url" value="http://localhost:8080/openidm/"/>
          <property name="user" value="openidm-admin"/>
          <property name="password" value="openidm-admin"/>
        </bean>
      </list>
    </property>
    <property name="resolverFactories">
      <list>
        <bean class="org.forgerock.openidm.workflow.activiti.impl.OpenIDMResolverFactory"></bean>
        <bean class="org.activiti.engine.impl.scripting.VariableScopeResolverFactory"></bean>
        <bean class="org.activiti.engine.impl.scripting.BeansResolverFactory"></bean>
      </list>
    </property>
    <property name="expressionManager">
      <bean class="org.forgerock.openidm.workflow.activiti.impl.OpenIDMExpressionManager"> </bean>
    </property>
   </bean>
```

5. Restart Tomcat.

```
$ cd /path/to/activiti/setup/
$ ant tomcat.start
```

6. Check that Activiti Explorer is running on localhost:9090/activiti-explorer, as you did in the previous section.

7. Configure OpenIDM to use the remote Activiti engine instead of the local, bundled Activiti engine.

   1. Copy the `workflow.json` configuration file to the OpenIDM configuration directory.

   ```
   $ cd /path/to/openidm/
   $ cp samples/misc/workflow.json conf/
   ```

   2. Edit the workflow configuration file to specify the remote Activiti engine.

   ```
   $ vi conf/workflow.json
   ```

   Ensure that the url, username, and password fields contain the values that correspond to your remote Activiti engine.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
   "enabled" : true,
   "location" : "remote",
   "engine" : {
       "url" : "http://localhost:9090/openidm-workflow-remote-2.1.2/",
       "username" : "youractivitiuser",
       "password" : "youractivitipassword"
   }
}
```

8. Start up OpenIDM.

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

9. Test the integration by sending the following CURL request to list the available workflows:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/processdefinition?_queryId=query-all-ids"
```

10. Log in to the Activiti Explorer of the remote Activiti engine (with the default username (kermit) and password (kermit).

11. Install the sample workflow (`example.bpmn20.xml`).

    1. In Activiti Explorer, click Manage.

    2. From the Deployments menu, select Upload New.

    3. Navigate to the sample workflow (`/path/to/openidm/samples /misc/example.bpmn20.xml)`

12. Verify the integration by sending the following CURL request:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/workflow/processinstance?_action=createProcessInstance"
 --data '{"_key":"osgiProcess"}'
```

The request should return a process ID, similar to the following:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
  "_id":"614",
  "processInstanceId":"614",
  "status":"ended",
  "businessKey":null,
  "processDefinitionId":"osgiProcess:7:603"
}
```

This request starts the osgiProcess and writes the list of installed workflows to the Tomcat server log file (`/path/to/activiti/apps/ apache-tomcat-6.0.32/logs/catalina.out`).

13. Test the integration with Activiti Explorer.
    1. Click Processes and select Osgi Process from the list of process definitions.
    2. Click Start Process.
    3. Check the output in the Tomcat server log file (`/path/to/activiti/apps/apache-tomcat-6.0.32/logs/catalina.out`. The list of installed workflows is written to the log file.

15.2.2. Configuring the Activiti Engine

Whether you use the embedded Activiti engine, or a remote Activiti engine, you configure the OpenIDM Activiti module in a file named `/path/to/openidm/conf/workflow.json`. If this file is absent from the configuration, the workflow module is unavailable for use. In the default OpenIDM installation, the `workflow.json` file assumes an embedded Activiti engine, and has the following configuration:

```
{
   "enabled"  : "true"
}
```

You can disable the workflow module by setting the "enabled" property in this file to "false".

A sample `workflow.json` file, with all configurable properties, is provided in `/path/to/openidm/samples/misc`. To configure an Activiti engine beyond the default configuration that is provided, edit this file as required and copy it to the `/path/to/openidm/conf` directory.

The sample `workflow.json` file contains the following configuration:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
    "enabled"  : "true",
    "location" : "remote",
    "engine" : {
        "url" : "http://localhost:9090/openidm-workflow-remote-2.1.2",
        "username" : "youractivitiuser",
        "password" : "youractivitipassword"
    },
    "mail" : {
        "host" : "yourserver.smtp.com",
        "port" : 587,
        "username" : "yourusername",
        "password" : "yourpassword",
        "starttls" : true
    },
    "history" : "audit"
}
```

These fields have the following meaning:

- `enabled`. Indicates whether the Activiti module is enabled for use. Possible values are `true` or `false`. The default value is `true`.
- `location`. Indicates whether the Activiti engine is embedded with OpenIDM, or remote. Possible values are `embedded` or `remote`. If `remote`, you must provide details for the `engine` property, below.
- `engine`. Specifies the details of the remote Activiti engine. The following fields must be defined:
  - `url`. The URL of the remote engine, including the host name and port number.
  - `username`. A user name for the remote Activiti engine.
  - `password`. The password for the user specified above.
- `mail`. Specifies the details of the mail server that Activiti will use to send email notifications. By default, Activiti uses the mail server `localhost:25`. To specify a different mail server, enter the details of the mail server here.
  - `host`. The host of the mail server.
  - `port`. The port number of the mail server.
  - `username`. The user name of the account that connects to the mail server.
  - `password`. The password for the user specified above.
  - `startTLS`. Whether startTLS should be used to secure the connection.
- `history`. Determines the history level that should be used for the Activiti engine. For more information, see Configuring the Activiti History Level.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

15.2.2.1. Configuring the Activiti History Level

The Activiti history level determines how much historical information is retained when workflows are executed. You can configure the history level by setting the `history` property in the `workflow.json` file, for example:

```
"history" : "audit"
```

The following history levels can be configured:

- `none`. No history archiving is done. This level results in the best performance for workflow execution, but no historical information is available.
- `activity`. Archives all process instances and activity instances. No details are archived.
- `audit`. This is the default level. All process instances, activity instances and submitted form properties are archived so that all user interaction through forms is traceable and can be audited.
- `full`. This is the highest level of history archiving and has the greatest performance impact. This history level stores all information as in the audit level as well as any process variable updates.

15.2.3. Defining Activiti Workflows

The following section outlines the process to follow when you create an Activiti workflow for OpenIDM. Before you start creating workflows, you must configure the Activiti engine, as described in Configuring the Activiti Engine.

1. Define your workflow in a text file, either using an editor, such as Activiti Eclipse BPMN 2.0 Designer, or a simple text editor.
2. Package the workflow definition file as a `.bar` file (Business Archive File). If you are using Eclipse to define the workflow, a `.bar` file is created when you select "Create deployment artifacts". Essentially, a `.bar` file is the same as a `.zip` file, but with the `.bar` extension.
3. Copy the `.bar` file to the `openidm/workflow` directory.
4. Invoke the workflow using a script (in `openidm/script/`) or directly using the REST interface. For more information, see Invoking Activiti Workflows.
5. You can also schedule the workflow to be invoked repeatedly, or at a future time.

15.2.4. Invoking Activiti Workflows

You can invoke workflows and business processes from any trigger point within OpenIDM, including reacting to situations discovered during reconciliation. Workflows can be invoked from script files, using the `openidm.action()` function, or directly from the REST interface.

The following sample script extract shows how to invoke a workflow from a script file:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
/*
 * Calling 'myWorkflow' workflow
 */

var params = {
 "foo" : "bar",
 "_key": "myWorkflow"
};

openidm.action('workflow/processinstance', {"_action" : "createProcessInstance"},
 {"keyrequired" : params});
```

You can invoke the same workflow from the REST interface by sending the following REST call to OpenIDM:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/workflow/processinstance?_action=createProcessInstance"
 --data '{"_key":"myWorkflow", "foo":"bar"}'
```

There are two ways in which you can specify the workflow definition that is used when a new workflow instance is started.

- `_key` specifies the `id` attribute of the workflow process definition, for example:

  ```
  <process id="sendNotificationProcess" name="Send Notification Process">
  ```

  If there is more than more than one workflow definition with the same `_key` parameter, the latest deployed version of the workflow definition is invoked.

- `_processDefinitionId` specifies the ID that is generated by the Activiti Process Engine when a workflow definition is deployed, for example:

  ```
  "sendNotificationProcess:1:104";
  ```

  You can obtain the `processDefinitionId` by querying the available workflows, for example:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
 "result": [
   {
     "name": "Process Start Auto Generated Task Auto Generated",
     "_id": "ProcessSAGTAG:1:728"
   },
   {
     "name": "Process Start Auto Generated Task Empty",
     "_id": "ProcessSAGTE:1:725"
   },
   ...
```

If you specify a `_key` and a `_processDefinitionId`, the `_processDefinitionId` is used because it is more precise.

You can use the optional `_businessKey` parameter to add specific business logic information to the workflow when it is invoked. For example, the following workflow invocation assigns the workflow a business key of `"newOrder"`. This business key can later be used to query "newOrder" processes.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/workflow/processinstance?_action=createProcessInstance"
 --data '{"_key":"myWorkflow", "_businessKey":"newOrder"}'
```

15.2.5. Querying Activiti Workflows

The Activiti implementation supports filtered queries that enable you to query the running process instances and tasks, based on specific query parameters. For example, the following query returns all process instances with the business key `"newOrder"`, as invoked in the previous section.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/workflow/processinstance?_queryId=filtered-query
 &businessKey=newOrder"
```

You can query process instances based on the value of any process instance variable by prefixing the variable name with `_var-`. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/workflow/processinstance?_queryId=filtered-query
 &_var-processvariablename=processvariablevalue"
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The standard Activiti properties can be queried using the Activiti notation, for example, `processDefinitionId=managedUserApproval:1:6405`. The query syntax applies to all queries with `_queryId=filtered-query`.

15.3. Managing Workflows Over the REST Interface

In addition to the queries described previously, the following examples show the endpoints that are exposed for managing workflows over the REST interface. The example output is based on the sample workflow that is provided in `openidm/samples/workflow`.

`openidm/workflow/processdefinition`

List the available workflow definitions, for example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/processdefinition?_queryId=query-all-ids"

{
  "result": [
    {
      "name": "Managed User Approval Workflow",
      "_id": "managedUserApproval:1:3"
    }
  ]
}
```

List the workflows, based on certain filter criteria, for example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/processdefinition?_queryId=
filtered-query&category=Examples"

{
  "result": [
    {
      "name": "Managed User Approval Workflow",
      "_id": "managedUserApproval:1:3"
    }
  ]
}
```

`openidm/workflow/processdefinition/{id}`

Obtain detailed information for a process definition, based on the ID. (The ID can be determined by querying all available process definitions). For example:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/processdefinition/contractorOnboarding:1:3"

{
  "key": "managedUserApproval",
  "_rev": "0",
  "formProperties": [],
  "category": "Examples",
  "_id": "managedUserApproval:1:3",
  "processDiagramResourceName": null,
  "description": null,
  "name": "Managed User Approval Workflow",
  "deploymentId": "1"
}
```

`openidm/workflow/processinstance`

Obtain the list of running workflows (process instances). The query returns a list of IDs. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/processinstance?_queryId=query-all-ids"

{
  "result": [
    {
      "processDefinitionId": "contractorOnboarding:1:3",
      "_id": "4"
    }
  ]
}
```

Obtain the list of running workflows based on specific filter criteria. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/processinstance?_queryId=
filtered-query&businessKey=myBusinessKey"
```

Start a workflow process instance. For example:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --data {"_key":"contractorOnboarding"}
 --request POST
 "http://localhost:8080/openidm/workflow/processinstance?_action=createProcessInstance"
```

`openidm/workflow/processinstance/{id}`

Obtain the details of the specified process instance. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/processinstance/4"

{
  "deleteReason": null,
  "processDefinitionId": "contractorOnboarding:1:3",
  "_rev": "0",
  "startTime": "2012-12-18T22:04:50.549+02:00",
  "startUserId": "user1",
  "_id": "4",
  "businessKey": null,
  "durationInMillis": null,
  "endTime": null,
  "superProcessInstanceId": null
}
```

Stop the specified process instance. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request DELETE
 "http://localhost:8080/openidm/workflow/processinstance/4"
```

`openidm/workflow/taskdefinition`

Query a task definition based on the process definition ID and the task name (`taskDefinitionKey`). For example:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/taskdefinition?_queryId=query-taskdefinition
&processDefinitionId=contractorOnboarding:1:6&taskDefinitionKey=decideApprovalTask"
{
  "dueDate": null,
  "taskCandidateGroup": [
    {
      "expressionText": "manager"
    }
  ],
  "formProperties": [
    {
      "type": {
        "values": {
          "accept": "Accept",
          "reject": "Reject"
...
```

`openidm/workflow/taskinstance`

Query all running task instances. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/taskinstance?_queryId=query-all-ids"

{
  "result": [
    {
      "name": "Contractor Approval",
      "_id": "70"
    }
  ]
}
```

Query task instances based on candidate users or candidate groups. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/taskinstance?_queryId=filtered-
query&taskCandidateUser=manager1"
```

or

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/taskinstance?_queryId=filtered-
query&taskCandidateGroup=management"
```

Note that you can include both users and groups in the same query.

openidm/workflow/taskinstance/{id}

Obtain detailed information for a running task, based on the task ID. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/workflow/taskinstance/70"

{
  "dueDate": null,
  "processDefinitionId": "contractorOnboarding:1:3",
  "owner": null,
  "taskDefinitionKey": "decideApprovalTask",
  "name": "Contractor Approval",
...
```

Update task-related data stored in the Activiti workflow engine. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --header "If-Match : *"
 --request PUT
 --data '{"description":"updated description"}'
 "http://localhost:8080/openidm/workflow/taskinstance/70"
```

Complete the specified task. The variables required by the task are provided in the request body. For example:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 --data '{"requestApproved":"true"}'
 "http://localhost:8080/openidm/workflow/taskinstance/70?_action=complete"
```

Claim the specified task. The ID of the user who claims the task is provided in the request body. For example:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 --data '{"userId":"manager1"}'
 "http://localhost:8080/openidm/workflow/taskinstance/70?_action=claim"
```

15.4. Example Activiti Workflows With OpenIDM

This section describes two example workflows - an email notification workflow, and a workflow that demonstrates provisioning, using the browser-based user interface.

15.4.1. Example Email Notification Workflow

This example uses the Activiti Eclipse BPMN 2.0 Designer to set up an email notification business process. The example relies on an SMTP server listening on `localhost`, port 25.

The example sets up a workflow that can accept parameters used to specify the sender and recipient of the mail.

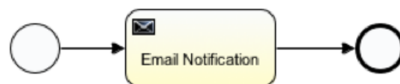
`${fromSender}`

Used to specify the sender

`${toEmail}`

Used to specify the recipient

Once you have defined the workflow, drag and drop components to create the workflow. This simple example uses only a `StartEvent`, `MailTask`, and `EndEvent`.



After creating the workflow, adjust the generated XML source code to use the variables inside the `<serviceTask>` tag shown in the following listing.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
 xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:activiti="http://activiti.org/bpmn"
 xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
 xmlns:omgdc="http://www.omg.org/spec/DD/20100524/DC"
 xmlns:omgdi="http://www.omg.org/spec/DD/20100524/DI"
 typeLanguage="http://www.w3.org/2001/XMLSchema"
 expressionLanguage="http://www.w3.org/1999/XPath"
 targetNamespace="http://www.activiti.org/test">
 <process id="EmailNotification" name="emailNotification">
   <documentation>Simple Email Notification Task</documentation>
   <startEvent id="startevent1" name="Start"></startEvent>
   <sequenceFlow id="flow1" name="" sourceRef="startevent1"
     targetRef="mailtask1"></sequenceFlow>
   <endEvent id="endevent1" name="End"></endEvent>
   <sequenceFlow id="flow2" name="" sourceRef="mailtask1"
     targetRef="endevent1"></sequenceFlow>
   <serviceTask id="mailtask1" name="Email Notification"
     activiti:type="mail">
     <extensionElements>
       <activiti:field name="to" expression="${toEmail}"
       ></activiti:field>
       <activiti:field name="from" expression="${fromSender}"
       ></activiti:field>
       <activiti:field name="subject" expression="Simple Email Notification"
       ></activiti:field>
       <activiti:field name="text">
         <activiti:expression><![CDATA[Here is a simple Email Notification
         from ${fromSender}.]]></activiti:expression>
       </activiti:field>
     </extensionElements>
   </serviceTask>
 </process>
 <bpmndi:BPMNDiagram id="BPMNDiagram_EmailNotification">
   <bpmndi:BPMNPlane bpmnElement="EmailNotification"
     id="BPMNPlane_EmailNotification">
     <bpmndi:BPMNShape bpmnElement="startevent1" id="BPMNShape_startevent1">
       <omgdc:Bounds height="35" width="35" x="170" y="250"></omgdc:Bounds>
     </bpmndi:BPMNShape>
     <bpmndi:BPMNShape bpmnElement="endevent1" id="BPMNShape_endevent1">
       <omgdc:Bounds height="35" width="35" x="410" y="250"></omgdc:Bounds>
     </bpmndi:BPMNShape>
     <bpmndi:BPMNShape bpmnElement="mailtask1" id="BPMNShape_mailtask1">
       <omgdc:Bounds height="55" width="105" x="250" y="240"></omgdc:Bounds>
     </bpmndi:BPMNShape>
     <bpmndi:BPMNEdge bpmnElement="flow1" id="BPMNEdge_flow1">
       <omgdi:waypoint x="205" y="267"></omgdi:waypoint>
       <omgdi:waypoint x="250" y="267"></omgdi:waypoint>
     </bpmndi:BPMNEdge>
     <bpmndi:BPMNEdge bpmnElement="flow2" id="BPMNEdge_flow2">
       <omgdi:waypoint x="355" y="267"></omgdi:waypoint>
       <omgdi:waypoint x="410" y="267"></omgdi:waypoint>
     </bpmndi:BPMNEdge>
   </bpmndi:BPMNPlane>
 </bpmndi:BPMNDiagram>
</definitions>
```

In Eclipse, select the project, then right click and select Create deployment artifacts to generate the components and package them in a .bar file for deployment in the `openidm/workflow` directory.

After you deploy the .bar, create a script named `openidm/script/triggerEmailNotification.js`. The script invokes the workflow.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
/*
 * Calling 'EmailNotification' workflow
 */

var params = {
 "_key" : "EmailNotification",
 "fromSender" : "noreply@openidm",
 "toEmail" : "jdoe@example.com"
};

openidm.action('workflow/processinstance', {"_action" : "createProcessInstance"},
{"keyrequired" : params});
```

You can also invoke the workflow over the REST interface with the following REST command:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --data '{"_key":"EmailNotification", "fromSender":"noreply@openidm", "toEmail":"jdoe@example.com"}'
 --request POST
 "http://localhost:8080/openidm/workflow/processinstance?_action=createProcessInstance"
```

To schedule the workflow to be invoked regularly, create a schedule configuration object named `openidm/conf/schedule-EmailNotification.json`. The following schedule invokes the workflow once per minute.

```
{
   "enabled" : true,
   "type" : "cron",
   "schedule" : "0 0/1 * * * ?",
   "invokeService" : "script",
   "invokeContext" : {
       "script" : {
           "type" : "text/javascript",
           "file" : "script/triggerEmailNotification.js"
       },
   }
}
```

15.4.2. Sample Workflow - Provisioning User Accounts

This example, provided in `openidm/samples/workflow`, uses workflows to provision user accounts. The example demonstrates the use of the browser-based user interface to manage workflows.

15.4.2.1. Overview of the Sample

The sample starts with a reconciliation process that loads user accounts from an XML file into the managed users repository. The reconciliation creates two users, with UIDs `user1` and `manager1`. Both users have the same password (`Welcome1`).

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The sample adds two new business roles to the configuration - `employee` (assigned to `user1`) and `manager` (assigned to `manager1`).

As part of the provisioning, employees are required to initiate a "Contract Onboarding" process. This process is a request to add a contractor to the managed users repository, with an option to include the contractor in the original data source (the XML file).

When the employee has completed the required form, the request is sent to the manager for approval. Any user with the role `"manager"` can claim the approval task. If the request is approved, the user is created in the managed users repository. If a request was made to add the user to the original data source (the XML file) this is done in a subsequent step.

The workflow uses embedded templates to build a more sophisticated input form. The form is validated with the server-side policy rules, described in *Using Policies to Validate Data*.

15.4.2.2. Running the Sample

1. Start OpenIDM with the configuration for the workflow sample.

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

2. Run reconciliation over the REST interface.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/recon?_action=recon&mapping=systemXmlfileAccounts_managedUser"
```

   Successful reconciliation returns an "_id" object, such as the following:

```
{"_id":"aea493f5-29ee-423d-b4b1-10449c60886c"}
```

   The two users are added to the repository. You can test this with the following REST query, which shows the two users, `manager1` and `user1`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

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

{
  "conversion-time-ms": 0,
  "result": [
    {
      "_rev": "0",
      "_id": "manager1"
    },
    {
      "_rev": "0",
      "_id": "user1"
    }
  ],
  "query-time-ms": 1
}
```

3. Log in to the user interface as `user1`, with password `Welcome1`. For information about logging in to the user interface, see *Overview of the Default User Interface*.

4. Under "Processes" click "Contractor onboarding process".

5. Complete the details of the new user, then click Start.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Contractor onboarding process

Contractor Details

Username: johnb
Email address: johnb@example.com
First Name: John
Last Name: Brand
Mobile Phone: 123456789
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,familyName

Provision to XML: Yes
Department: Finance
Job Title: Contract Accountant

6. Log out of the UI.
7. Log in to the UI as `manager1`, with password `Welcome1`.
8. Under "Tasks that are in my group's queue" click "Contractor Approval".
9. From the drop-down list, select "Assign to me".

   Note that the "Contractor Approval" task has now moved under "My tasks".
10. Under "My tasks" click "Contractor Approval".
11. Under Actions, click Details.

    The form containing the details of the contractor is displayed.
12. At the bottom of the form, select a decision from the drop-down list (either "Accept" or "Reject"), then click Complete.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

If you Accept the new contractor details, the user account is created in the repository. You can check the new account by running the following REST command:

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

{
  "conversion-time-ms": 0,
  "result": [
    {
      "_rev": "0",
      "_id": "manager1"
    },
    {
      "_rev": "0",
      "_id": "user1"
    },
    {
      "_rev": "0",
      "_id": "51afe0f8-94c3-45c5-8c69-319e6ef5981f"
    }
  ],
  "query-time-ms": 1
}
```

Display the details of the new user, by running a REST query on the user ID, as follows:

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request GET
 "http://localhost:8080/openidm/managed/user/51afe0f8-94c3-45c5-8c69-319e6ef5981f"

{
  "city": "",
  "country": "",
  "address2": "",
  "address1": "",
  "lastPasswordAttempt": "Fri Dec 14 2012 13:54:02 GMT+0200 (SAST)",
  "passwordAttempts": "0",
  "stateProvince": "",
  "postalCode": "",
  "lastPasswordSet": "",
  "jobTitle": "Accountant",
  "department": "Finance",
  "manager": "user1",
  "familyName": "Doe",
  "givenName": "John",
  "userName": "johnd",
  "_rev": "0",
  "_id": "51afe0f8-94c3-45c5-8c69-319e6ef5981f",
  "phoneNumber": "123456789",
  "email": "johnd@example.com",
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
  "startDate": "12/12/2012",
  "endDate": "12/12/2012",
  "description": "Contract accountant",
  "provisionToXML": "1",
  "accountStatus": "active",
  "roles": "openidm-authorized"
}
```

You can now log in to the UI as the new user (with the details that you specified in Step 5). Under "Notifications" you will see a welcome message indicating the working dates of the new user. If you log in as `user1` you are notified of the result of the manager's decision.

If you specified that the new user should be added to the original data source, you will see that the account was added to the XML file:

```
$ cd /path/to/openidm
$ cat samples/workflow/data/xmlConnectorData.xml
...
   >ri:__ACCOUNT__<
      >icf:__DESCRIPTION__<Contract accountant>/icf:__DESCRIPTION__<
      >ri:roles<openidm-authorized>/ri:roles<
      >ri:mobileTelephoneNumber<123456789>/ri:mobileTelephoneNumber<
      >ri:firstname<John>/ri:firstname<
      >ri:manager<user1>/ri:manager<
      >ri:startDate<12/12/2012>/ri:startDate<
      >ri:jobTitle<Accountant>/ri:jobTitle<
      >icf:__UID__<201e0d50-3313-47b3-9bd1-30c1c7dd1cee>/icf:__UID__<
      >icf:__NAME__<johnd>/icf:__NAME__<
      >ri:email<johnd@example.com>/ri:email<
      >icf:__PASSWORD__<MyPassw0rd>/icf:__PASSWORD__<
      >ri:department<Finance>/ri:department<
      >ri:endDate<12/12/2012>/ri:endDate<
      >ri:lastname<Doe>/ri:lastname<
   >/ri:__ACCOUNT__<
...
```

If you declined the approval request, the user will not be created in either data source.

You can see the details of the workflow definition in `samples/workflow/workflow/contractorOnboarding.bpmn20.xml`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 16 Using Audit Logs

OpenIDM auditing can publish and log all relevant system activity to the targets you specify. Auditing can include data from reconciliation as a basis for reporting, access details, and activity logs that capture operations on internal (managed) objects and external (system) objects. Auditing provides the data for all the relevant reports, including orphan account reports.

The auditing interface allows you to push auditing data to files and to the OpenIDM repository.

16.1. Audit Log Types

This section describes the types of audit log OpenIDM provides.

Access Log

OpenIDM writes messages concerning access to the REST API in this log.

Default file: `openidm/audit/access.csv`

Activity Log

OpenIDM logs operations on internal (managed) and external (system) objects to this log type.

Entries in the activity log contain identifiers both for the reconciliation or synchronization action that triggered the activity, and also for the original caller and the relationships between related actions.

Default file: `openidm/audit/activity.csv`

Reconciliation Log

OpenIDM logs the results of a reconciliation run, including situations and the resulting actions taken to this log type. The activity log contains details about the actions, where log entries display parent activity identifiers, `recon/reconID`.

Default file: `openidm/audit/recon.csv`

Where an action happens in the context of a higher level business function, the log entry points to a parent activity for the context. The relationships are hierarchical. For example, a synchronization operation could result from scheduled reconciliation for an object type. OpenIDM also logs the top level root activity with each entry, making it possible to query related activities.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

16.2. Audit Log File Formats

This section describes the audit log file formats to help you map these to the reports you generate.

Access Log Fields

Access messages are split into the following fields.

`"_id"`

UUID for the message object, such as `"0419d364-1b3d-4e4f-b769-555c3ca098b0"`

`"action"`

Action requested, such as `"authenticate"`

`"ip"`

IP address of the client. For access from the local host, this can appear for example as `"0:0:0:0:0:0:0:1%0"`.

`"principal"`

Principal requesting the operation, such as `"openidm-admin"`

`"roles"`

Roles associated with the principal, such as `"[openidm-admin, openidm-authorized]"`

`"status"`

Result of the operation, such as `"SUCCESS"`

`"timestamp"`

Time when OpenIDM logged the message, in UTC format, for example `"2012-11-18T08:48:00.160Z"`

Activity Log Fields

Activity messages are split into the following fields.

`"_id"`

UUID for the message object, such as `"0419d364-1b3d-4e4f-b769-555c3ca098b0"`

`"action"`

Action performed, such as `"create"`. See the section on *Event Types* for a list.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`"activityId"`

UUID for the activity corresponding to the UUID of the resource context

`"after"`

JSON representation of the object resulting from the activity

`"before"`

JSON representation of the object prior to the activity

`"message"`

Human readable text about the activity

`"objectId"`

Object identifier such as `"managed/user/DDOE1"`

`"parentActionId"`

UUID of the action leading to the activity

`"requester"`

Principal requesting the operation

`"rev"`

Object revision number

`"rootActionId"`

UUID of the root cause for the activity. This matches a corresponding `"rootActionId"` in a reconciliation message.

`"status"`

Result of the operation, such as `"SUCCESS"`

`"timestamp"`

Time when OpenIDM logged the message, in UTC format, for example `"2012-11-18T08:48:00.160Z"`

Reconciliation Log Fields

Reconciliation messages are split into the following fields.

`"_id"`

UUID for the message object, such as `"0419d364-1b3d-4e4f-b769-555c3ca098b0"`

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`"action"`

Synchronization action, such as `"CREATE"`. See the section on *Actions* for a list.

`"ambiguousTargetObjectIds"`

When the situation is AMBIGUOUS or UNQUALIFIED and OpenIDM cannot distinguish between more than one target object, OpenIDM logs the identifiers of the objects in this field in comma-separated format. This makes it possible to figure out what was ambiguous afterwards.

`"entryType"`

Kind of reconciliation log entry, such as `"start"`, or `"summary"`.

`"message"`

Human readable text about the reconciliation action

`"reconciling"`

What OpenIDM is reconciling, `"source"` for the first phase, `"target"` for the second phase

`"reconId"`

UUID for the reconciliation operation, which is the same for all entries pertaining to the reconciliation run.

`"rootActionId"`

UUID of the root cause for the activity. This matches a corresponding `"rootActionId"` in an activity message.

`"situation"`

The situation encountered. See the section describing synchronization situations for a list.

`"sourceObjectId"`

UUID for the source object.

`"status"`

Result of the operation, such as `"SUCCESS"`

`"targetObjectId"`

UUID for the target object

`"timestamp"`

Time when OpenIDM logged the message, in UTC format, for example `"2012-11-18T08:48:00.160Z"`

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

16.3. Audit Configuration

OpenIDM exposes the audit logging configuration under `http://localhost:8080/openidm/config/audit` for the REST API, and in the file `conf/audit.json` where you installed OpenIDM. The default `conf/audit.json` file contains the following object.

```
{
    "eventTypes": {
        "activity": {
            "filter": {
                "actions": [
                    "create",
                    "update",
                    "delete",
                    "patch",
                    "action"
                ]
            },
            "watchedFields" : [ ],
            "passwordFields" : [ "password" ]
        },
        "recon": {}
    },
    "logTo": [
        {
            "logType": "csv",
            "location": "audit",
            "recordDelimiter": ";"
        },
        {
            "logType": "repository"
        }
    ]
}
```

16.3.1. Event Types

The `eventTypes` configuration specifies what events OpenIDM writes to audit logs. OpenIDM supports two `eventTypes`: `activity` for the activity log, and `recon` for the reconciliation log. The filter for actions under activity logging shows the actions on managed or system objects for which OpenIDM writes to the activity log.

The `filter actions` list enables you to configure the conditions that result in actions being written to the activity log.

`read`

When an object is read by using its identifier.

`create`

When an object is created.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`update`

When an object is updated.

`delete`

When an object is deleted.

`patch`

When an object is partially modified.

`query`

When a query is performed on an object.

`action`

When an action is performed on an object.

You can optionally add a `filter triggers` list that specifies the actions that are logged for a particular trigger. For example, the following addition to the `audit.json` file specifies that only `create` and `update` actions are logged for an activity that was triggered by a `recon`.

```
...
         "filter" : {
             "actions" : [
                 "create",
                 "update",
                 "delete",
                 "patch",
                 "action"
             ],
             "triggers" : {
                 "recon" : [
                     "create",
                     "update"
                 ]
             }
         },
         "watchedFields" : [ ],
...
```

If a trigger is provided, but no actions are specified, nothing is logged for that trigger. If a trigger is omitted, all actions are logged for that trigger. In the current OpenIDM release, only the `recon` trigger is implemented. For a list of reconciliation actions that can be logged, see *Synchronization Actions*.

The `watchedFields` parameter enables you to specify a list of fields that should be "watched" for changes. When the value of one of the fields in this list changes, the change is logged in the audit log, under the column `"changedFields"`. Fields are listed in comma-separated format, for example:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
"watchedFields" : [ "email", "address" ]
```

The `passwordFields` parameter enables you to specify a list of fields that are considered passwords. This parameter functions much like the `watchedFields` parameter in that changes to these field values are logged in the audit log, under the column `"changedFields"`. In addition, when a password field is changed, the boolean `"passwordChanged"` flag is set to `true` in the audit log. Fields are listed in comma-separated format, for example:

```
"passwordFields" : [ "password", "username" ]
```

The `logTo` list enables you to specify where OpenIDM writes the log.

`csv`

Write to a comma-separated variable format file in the location specified, relative to the directory in which you installed OpenIDM.

Audit file names are fixed, `access.csv`, `activity.csv`, and `recon.csv`.

The `"recordDelimiter"` property enables you to specify the separator between each record.

`repository`

Write to the OpenIDM database repository.

OpenIDM stores entries under the `/openidm/repo/audit/` context. Such entries appear as `audit/access/_id`, `audit/activity/_id`, and `audit/recon/_id`, where the *_id* is the UUID of the entry, such as `0419d364-1b3d-4e4f-b769-555c3ca098b0`.

In the OrientDB repository, OpenIDM stores records in `audit_access`, `audit_activity`, and `audit_recon` tables.

In a JDBC repository, OpenIDM stores records in `auditaccess`, `auditactivity`, and `auditrecon` tables.

16.4. Generating Reports

When generating reports from audit logs, you can correlate information from activity and reconciliation logs by matching the `"rootActionId"` on entries in both logs.

The following MySQL query shows a join of the audit activity and audit reconciliation tables using root action ID values.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
mysql> select distinct auditrecon.activity,auditrecon.sourceobjectid,
 auditrecon.targetobjectid,auditactivity.activitydate,auditrecon.status
 from auditactivity inner join auditrecon
 auditactivity.rootactionid=auditrecon.rootactionid
 where auditrecon.activity is not null group by auditrecon.sourceobjectid;
+----------+-------------------------+---------------------+---------------------+---------+
| activity | sourceobjectid          | targetobjectid      | activitydate        | status  |
+----------+-------------------------+---------------------+---------------------+---------+
| CREATE   | system/xmlfile/account/1 | managed/user/juser   | 2012-01-17T07:59:12 | SUCCESS |
| CREATE   | system/xmlfile/account/2 | managed/user/ajensen | 2012-01-17T07:59:12 | SUCCESS |
| CREATE   | system/xmlfile/account/3 | managed/user/bjensen | 2012-01-17T07:59:12 | SUCCESS |
+----------+-------------------------+---------------------+---------------------+---------+
3 rows in set (0.00 sec)
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 17
Sending Email

This chapter shows you how to configure the outbound email service, so that you can send email through OpenIDM either by script or through the REST API.

Procedure 17.1. To Set Up Outbound Email

The outbound email service relies on a configuration object to identify the email account used to send messages.

1. Shut down OpenIDM.
2. Copy the sample configuration to `openidm/conf`.

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

3. Edit `external.email.json` to reflect the account used to send messages.

```
{
        "host" : "smtp.example.com",
        "port" : "25",
        "username" : "openidm",
        "password" : "secret12",
        "mail.smtp.auth" : "true",
        "mail.smtp.starttls.enable" : "true"
}
```

OpenIDM encrypts the password you provide.

Follow these hints when editing the configuration.

`"host"`

SMTP server host name or IP address. This can be `"localhost"` if the server is on the same system as OpenIDM.

`"port"`

SMTP server port number such as 25, or 587

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`"username"`

Mail account user name needed when `"mail.smtp.auth" : "true"`

`"password"`

Mail account user password needed when `"mail.smtp.auth" : "true"`

`"mail.smtp.auth"`

If `"true"`, use SMTP authentication

`"mail.smtp.starttls.enable"`

If `"true"`, use TLS

`"from"`

Optional default `From:` address

4. Start up OpenIDM.
5. Check that the email service is active.

```
-> scr list
...
[   6] [active       ] org.forgerock.openidm.external.email
...
```

17.1. Sending Mail Over REST

Although you are more likely to send mail from a script in production, you can send email using the REST API by sending an HTTP POST to `/openidm/external/email` in order to test that your configuration works. You pass the message parameters as POST parameters, URL encoding the content as necessary.

The following example sends a test email using the REST API.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST
 "http://localhost:8080/openidm/external/email?
 _from=openidm@example.com&_to=admin@example.com&
 _subject=Test&_body=Test"
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

17.2. Sending Mail From a Script

You can send email from using the resource API functions with the `external/email` context, as in the following example, where `params` is an object containing the POST parameters.

```
var params =  new Object();
params._from = "openidm@example.com";
params._to = "admin@example.com";
params._cc = "wally@example.com,dilbert@example.com";
params._subject = "OpenIDM recon report";
params._type = "text/html";
params._body = "<html><body><p>Recon report follows...</p></body></html>";

openidm.action("external/email", params);
```

OpenIDM supports the following POST parameters.

`_from`

Sender mail address

`_to`

Comma-separated list of recipient mail addresses

`_cc`

Optional comma-separated list of copy recipient mail addresses

`_bcc`

Optional comma-separated list of blind copy recipient mail addresses

`_subject`

Email subject

`_body`

Email body text

`_type`

Optional MIME type. One of `"text/plain"`, `"text/html"`, or `"text/xml"`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 18
OpenIDM Project Best Practices

This chapter lists points to check when implementing an identity management solution with OpenIDM.

18.1. Implementation Phases

Any identity management project should follow a set of well defined phases, where each phase defines discrete deliverables. The phases take the project from initiation to finally going live with a tested solution.

18.1.1. Initiation

The project's initiation phase involves identifying and gathering project background, requirements, and goals at a high level. The deliverable for this phase is a statement of work or a mission statement.

18.1.2. Definition

In the definition phase, you gather more detailed information on existing systems, determine how to integrate, describe account schemas, procedures, and other information relevant to the OpenIDM deployment. The deliverable for this phase is one or more documents that define detailed requirements for the project, and that cover project definition, the business case, use cases to solve, and functional specifications.

The definition phase should capture at least the following.

User Administration and Management

Procedures for managing users and accounts, who manages users, what processes look like for joiners, movers and leavers, and what is required of OpenIDM to manage users

Password Management and Password Synchronization

Procedures for managing account passwords, password policies, who manages passwords, and what is required of OpenIDM to manage passwords

Security Policy

What security policies defines for users, accounts, passwords, and access control

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Target Systems

Target systems and resources with which OpenIDM must integrate. Information such as schema, attribute mappings and attribute transformation flow, credentials and other integration specific information.

Entitlement Management

Procedures to manage user access to resources, individual entitlements, grouping provisioning activities into encapsulated concepts such as roles and groups

Synchronization and Data Flow

Detailed outlines showing how identity information flows from authoritative sources to target systems, attribute transformations required

Interfaces

How to secure the REST, user and file-based interfaces, and to secure the communication protocols involved

Auditing and Reporting

Procedures for auditing and reporting, including who takes responsibility for auditing and reporting, and what information is aggregated and reported. Characteristics of reporting engines provided, or definition of the reporting engine to be integrated.

Technical Requirements

Other technical requirements for the solution such as how to maintain the solution in terms of monitoring, patch management, availability, backup, restore and recovery process. This includes any other components leveraged such as a ConnectorServer and plug-ins for password synchronization on Active Directory, or OpenDJ.

18.1.3. Design

This phase focuses on solution design including on OpenIDM and other components. The deliverables for this phase are the architecture and design documents, and also success criteria with detailed descriptions and test cases to verify when project goals have been met.

18.1.4. Build

This phase builds and tests the solution prior to moving the solution into production.

18.1.5. Production

This phase deploys the solution into production until an application steady state is reached and maintenance routines and procedures can be applied.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Chapter 19
Troubleshooting

When things are not working check this chapter for tips and answers.

19.1. OpenIDM Stopped in Background

When you start OpenIDM in the background without having disabled the text console, the job can stop immediately after startup.

```
$ ./startup.sh &
[2] 346
$ ./startup.sh
Using OPENIDM_HOME:   /path/to/openidm
Using OPENIDM_OPTS:   -Xmx1024m
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
->

[2]+  Stopped                 ./startup.sh
```

To resolve this problem, make sure you remove `openidm/bundle/org.apache.felix.shell.tui-1.4.1.jar` before starting OpenIDM, and also remove Felix cache files in `openidm/felix-cache/`.

19.2. Internal Server Error During Reconciliation or Synchronization

You might see an error message such as the following returned from reconciliation or synchronization.

```
{
    "error": "Conflict",
    "description": "Internal Server Error:
        org.forgerock.openidm.sync.SynchronizationException:
        Cowardly refusing to perform reconciliation with an
        empty source object set: Cowardly refusing to perform
        reconciliation with an empty source object set"
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

This error can be misleading. This usually means the connector is not able to communicate with the target source.

Check the settings for your connector. For example, with the XML connector you get this error if the filename for the source is invalid. With the LDAP connector, you can get this error if your connector cannot contact the target LDAP server.

19.3. The scr list Command Shows Sync Service As Unsatisfied

You might encounter this message in the logs.

```
WARNING: Loading configuration file /path/to/openidm/conf/sync.json failed
org.forgerock.openidm.config.InvalidException:
 Configuration for org.forgerock.openidm.sync could not be parsed and may not
     be valid JSON : Unexpected character ('}' (code 125)): expected a value
     at [Source: java.io.StringReader@3951f910; line: 24, column: 6]
 at org.forgerock.openidm.config.crypto.ConfigCrypto.parse...
 at org.forgerock.openidm.config.crypto.ConfigCrypto.encrypt...
 at org.forgerock.openidm.config.installer.JSONConfigInstaller.setConfig...
```

This indicates a syntax error in `openidm/conf/sync.json`. After fixing your configuration, change to the `/path/to/openidm/` directory, and use the **cli.sh validate** command to check that your configuration files are valid.

```
$ cd /path/to/openidm ; ./cli.sh validate
Using boot properties at /path/to/openidm/conf/boot/boot.properties
...................................................................
[Validating] Load JSON configuration files from:
[Validating]  /path/to/openidm/conf
[Validating] audit.json .................................. SUCCESS
[Validating] authentication.json ......................... SUCCESS
[Validating] managed.json ................................ SUCCESS
[Validating] provisioner.openicf-xml.json ................ SUCCESS
[Validating] repo.orientdb.json .......................... SUCCESS
[Validating] router.json ................................. SUCCESS
[Validating] scheduler-reconcile_systemXmlAccounts_managedUser.json  SUCCESS
[Validating] sync.json ................................... SUCCESS
```

19.4. JSON Parsing Error

You might encounter this error message in the logs.

```
"Configuration for org.forgerock.openidm.provisioner.openicf could not be
 parsed and may not be valid JSON : Unexpected character ('}' (code 125)):
 was expecting double-quote to start field name"
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The error message usually points precisely to the point where the JSON file has the syntax problem. The error above was caused by a excess comma in the JSON file, `{"attributeName":{},{},}`. The second comma is too much.

The situation usually results in the service the JSON file configures being left in the `unsatisfied` state.

After fixing your configuration, change to the `/path/to/openidm/` directory, and use the **cli.sh validate** command to check that your configuration files are valid.

19.5. System Not Available

OpenIDM throws the following error as a result of a reconciliation where the source systems configuration can not be found.

```
{
    "error": "Conflict",
    "description": "Internal Server Error:
        org.forgerock.openidm.sync.SynchronizationException:
        org.forgerock.openidm.objset.ObjectSetException:
        System: system/HR/account is not available.:
        org.forgerock.openidm.objset.ObjectSetException:
        System: system/HR/account is not available.:
        System: system/HR/account is not available."
}
```

This error occurs when the `"name"` property value in `provisioner.resource.json` is changed from `HR` to something else.

The same error also occurs when a provisioner configuration fails to load due to misconfiguration, or when the path to the data file for a CSV or XML connector is incorrectly set.

19.6. Bad Connector Host Reference in Provisioner Configuration

You might see the following error when a provision configuration loads.

```
Wait for meta data for config org.forgerock.openidm.provisioner.openicf-scriptedsql
```

In this case the configuration fails to load because some information is missing. One possible cause is a wrong value for `connectorHostRef` in the provisioner configuration file.

For local Java connector servers, the following rules apply.

- If the connector .jar is installed as a bundle under `openidm/bundle`, then the value must be `"connectorHostRef" : "osgi:service/org.forgerock.openicf.framework.api.osgi.ConnectorManager",`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

- If the connector .jar is installed as a connector under `openidm/connectors`, then the value must be `"connectorHostRef" : "#LOCAL",`.

19.7. Missing Name Attribute

In this case, the situation in the audit recon log shows "NULL".

A missing name attribute error, followed by an `IllegalArgumentException`, points to misconfiguration of the correlation rule, with the correlation query pointing to the external system. Such queries usually reference the "name" field which, if empty, leads to the error below.

```
Jan 20, 2012 1:59:58 PM
 org.forgerock.openidm.provisioner.openicf.commons.AttributeInfoHelper build
SEVERE: Failed to build name attribute out of [null]
Jan 20, 2012 1:59:58 PM
 org.forgerock.openidm.provisioner.openicf.impl.OpenICFProvisionerService query
SEVERE: Operation [query, system/ad/account] failed with Exception on system
 object: java.lang.IllegalArgumentException: Attribute value must be an
 instance of String.
Jan 20, 2012 1:59:58 PM org.forgerock.openidm.router.JsonResourceRouterService
 handle
WARNING: JSON resource exception
org.forgerock.json.resource.JsonResourceException: IllegalArgumentException
 at org.forgerock.openidm.provisioner....OpenICFProvisionerService.query...
 at org.forgerock.openidm.provisioner.....OpenICFProvisionerService.handle...
 at org.forgerock.openidm.provisioner.impl.SystemObjectSetService.handle...
 at org.forgerock.json.resource.JsonResourceRouter.handle...
```

Check your `correlationQuery`. Another symptom of a broken correlationQuery is that the audit recon log shows a situation of "NULL", and no onCreate, onUpdate or similar scripts are executed.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Appendix A. File Layout

When you unpack and start OpenIDM 2.1, you create the following files and directories. Note that the precise paths will depend on the install, project, and working directories that you have selected during startup. For more information, see *Specifying the OpenIDM Startup Configuration*.

`openidm/audit/`

OpenIDM audit log directory default location, created at run time, as configured in `openidm/conf/audit.json`

`openidm/audit/access.csv`

Default OpenIDM access audit log

`openidm/audit/activity.csv`

Default OpenIDM activity audit log

`openidm/audit/recon.csv`

Default OpenIDM reconciliation audit log

`openidm/bin/`

OpenIDM core libraries and scripts

`openidm/bin/create-openidm-logrotate.sh`

Script to create an `openidmlog` log rotation scheduler for inclusion under `/etc/logrotate.d/`

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`openidm/bin/create-openidm-rc.sh`

Script to create an `openidm` resource definition file for inclusion under `/etc/init.d/`

`openidm/bin/defaults/script`

Default scripts required to run specific services. In general, you should not modify these scripts. Instead, add customized scripts to the `openidm/script` folder.

`openidm/bin/defaults/script/info/login.js`

Provides information about the current OpenIDM session.

`openidm/bin/defaults/script/info/ping.js`

Provides basic information about the health of an OpenIDM system

`openidm/bin/defaults/script/policy.js`

Defines each policy and specifies how policy validation is performed

`openidm/bin/defaults/script/policyFilter.js`

Enforces policy validation

`openidm/bin/defaults/script/router-authz.js`

Provides the functions that enforce access rules

`openidm/bin/defaults/script/ui/*`

Scripts required by the UI

`openidm/bin/defaults/script/workflow/*`

Default workflow scripts

`openidm/bin/felix.jar`
`openidm/bin/openidm.jar`
`openidm/bin/org.apache.felix.gogo.runtime-0.10.0.jar`
`openidm/bin/org.apache.felix.gogo.shell-0.10.0.jar`

Files relating to the Apache Felix OSGi framework

`openidm/bin/launcher.bat`
`openidm/bin/launcher.jar`
`openidm/bin/launcher.json`

Files relating to the startup configuration

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`openidm/bin/LICENSE.TXT`
`openidm/bin/NOTICE.TXT`

Files relating to the Apache Software License

`openidm/bin/MonitorService.bat`
`openidm/bin/prunmgr.exe`
`openidm/bin/amd64/prunsrv.exe`
`openidm/bin/i386/prunsrv.exe`
`openidm/bin/ia64/prunsrv.exe`

Files required by the user interface to monitor and configure installed services

`openidm/bin/startup/`
`openidm/bin/startup/OS X - Run OpenIDM In Background.command`
`openidm/bin/startup/OS X - Run OpenIDM In Terminal Window.command`
`openidm/bin/startup/OS X - Stop OpenIDM.command`

Clickable commands for Mac OS X

`openidm/bin/workflow/`

Files related to the Activiti workflow engine

`openidm/bundle/`

OSGi bundles and modules required by OpenIDM. Upgrade can install new and upgraded bundles here.

`openidm/cli.bat`
`openidm/cli.sh`

Management commands for operations such as validating configuration files

`openidm/conf/`

OpenIDM configuration files, including .properties files and JSON files. You can also access JSON views through the REST interface.

`openidm/conf/audit.json`

Audit event publisher configuration file

`openidm/conf/authentication.json`

Authentication configuration file for access to the REST API

`openidm/conf/boot/boot.properties`

OpenIDM bootstrap properties

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`openidm/conf/config.properties`

Felix and OSGi bundle configuration properties

`openidm/conf/endpoint-*.json`

Endpoint configuration files required by the UI for the default workflows

`openidm/conf/jetty.xml`

Jetty configuration controlling access to the REST interface

`openidm/conf/logging-config.xml`

Experimental log configuration

`openidm/conf/logging.properties`

OpenIDM log configuration properties

`openidm/conf/managed.json`

Managed object configuration file

`openidm/conf/policy.json`

Default policy configuration

`openidm/conf/process-access.json`

Workflow access configuration

`openidm/conf/repo.orientdb.json`

OrientDB internal repository configuration file

`openidm/conf/router.json`

Router service configuration file

`openidm/conf/scheduler.json`

Scheduler service configuration

`openidm/conf/system.properties`

System configuration properties used when starting OpenIDM services

`openidm/conf/ui-configuration.json`

Main configuration file for the browser-based user interface

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`openidm/conf/ui-countries.json`

Configurable list of countries available when registering users in the user interface

`openidm/conf/ui-secquestions.json`

Configurable list of security questions available when registering users in the user interface

`openidm/conf/workflow.json`

Configuration of the Activiti workflow engine

`openidm/connectors/`

OpenICF connector libraries. OSGi enabled connector libraries can also be stored in `openidm/bundle/`.

`openidm/db/`

Internal repository files, including both OrientDB files and data definition language scripts for JDBC based repositories such as MySQL

`openidm/felix-cache/`

Bundle cache directory created when the Felix framework is started

`openidm/logs/`

OpenIDM service log directory

`openidm/logs/openidm0.log.*`

OpenIDM service log files as configured in `openidm/conf/logging.properties`

`openidm/samples/`

OpenIDM sample configurations

`openidm/samples/customendpoint`

Sample custom endpoint configuration. For more information, see *Adding Custom Endpoints*.

`openidm/samples/infoservice`

Sample that shows how to use the configurable information service. For more information, see *Obtaining Information About an OpenIDM Instance*.

`openidm/samples/misc/`

Sample configuration files

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`openidm/samples/openam/`

Sample that shows how to protect OpenIDM with OpenAM

`openidm/samples/provisioners/`

Sample connector configuration files

`openidm/samples/sample1/`

XML file connector sample

`openidm/samples/sample2/`

OpenDJ connector sample with no back link

`openidm/samples/sample2b/`

OpenDJ connector sample with back link

`openidm/samples/sample2c/`

OpenDJ connector sample synchronizing users' LDAP group membership

`openidm/samples/sample2d/`

OpenDJ connector sample synchronizing LDAP groups

`openidm/samples/sample3/`

Scripted SQL connector sample for MySQL

`openidm/samples/sample4/`

CSV connector sample

`openidm/samples/sample5/`

LDAP to OpenIDM to Active Directory attribute flow sample using XML resources rather than actual directories

`openidm/samples/sample6/`

LiveSync sample for use with one or two LDAP servers

`openidm/samples/sample7/`

Sample exposing identities with a SCIM-line schema

`openidm/samples/sample8/`

Sample demonstrating logging in scripts

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`openidm/samples/sample9/`

Sample showing asynchronous reconciliation with workflows

`openidm/samples/schedules/`

Sample schedule configuration files

`openidm/samples/security/`

Sample key store, trust store, and certificates

`openidm/samples/taskscanner/`

Sample sunset scanning task. For more information, see *Scanning Data to Trigger Tasks*.

`openidm/samples/workflow/`

Typical use case of a workflow for provisioning

`openidm/script/`

OpenIDM location for JavaScript files referenced in the configuration

`openidm/script/access.js`

Default authorization policy script

`openidm/security/`

OpenIDM security configuration, key store, and trust store

`openidm/shutdown.sh`

Script to shutdown OpenIDM services based on the process identifier

`openidm/startup.bat`

Script to start OpenIDM services on Windows

`openidm/startup.sh`

Script to start OpenIDM services on UNIX

`openidm/ui/`

OpenIDM graphical UI files

`openidm/workflow/`

OpenIDM location for BPMN 2.0 workflows and .bar files

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Appendix B. Ports Used

By default the OpenIDM 2.1 Jetty configuration in `openidm/conf/jetty.xml` specifies the following ports.

`8080`

HTTP access to the REST API, requiring OpenIDM authentication. This port is not secure, exposing clear text passwords and all data that is not encrypted. This port is therefore not suitable for production use.

`8443`

HTTPS access to the REST API, requiring OpenIDM authentication

`8444`

HTTPS access to the REST API, requiring SSL mutual authentication. Clients presenting certificates found in the trust store under `openidm/security/` are granted access to the system.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Appendix C. Data Models and Objects Reference

OpenIDM allows you to customize a variety of objects that can be addressed via a URL or URI, and that have a common set of functions that OpenIDM can perform on them such as CRUD, query, and action.

Depending on how you intend to use them, different objects are appropriate.

Table C.1. OpenIDM Objects

Object Type	Intended Use	Special Functionality
Managed objects	Serve as targets and sources for synchronization, and to build virtual identities.	Provide appropriate auditing, script hooks, declarative mappings and so forth in addition to the REST interface.
Configuration objects	Ideal for look-up tables or other custom configuration, which can be configured externally like any other system configuration.	Adds file view, REST interface, and so forth
Repository objects	The equivalent of arbitrary database table access. Appropriate for managing data purely through the underlying data store or repository API.	Persistence and API access
System objects	Representation of target resource objects, such as accounts, but also resource objects such as groups.	

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Object Type	Intended Use	Special Functionality
Audit objects	Houses audit data in the OpenIDM internal repository.	
Links	Defines a relation between two objects.	

C.1. Managed Objects

A *managed object* in OpenIDM is an object which represents the identity-related data managed by OpenIDM. Managed objects are stored by OpenIDM in its data store. All managed objects are JSON-based data structures.

C.1.1. Managed Object Schema

Managed objects have an associated schema to enforce a specific data structure. Schema is specified using the JSON Schema specification. This is currently an Internet-Draft, with implementations in multiple programming languages.

C.1.1.1. Managed Object Reserved Properties

Top-level properties in a managed object that begin with an underscore (`_`) are reserved by OpenIDM for internal use, and are not explicitly part of its schema. Internal properties are read-only, and are ignored when provided by the REST API client.

The following properties exist for all managed objects in OpenIDM.

_id

string

The unique identifier for the object. This value forms a part of the managed object's URI.

_rev

string

The revision of the object. This is the same value that is exposed as the object's ETag through the REST API. The content of this attribute is not defined. No consumer should make any assumptions of its content beyond equivalence comparison. This attribute may be provided by the underlying data store.

_schema_id

string

The a reference to the schema object that the managed object is associated with.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

_schema_rev

string

The revision of the schema that was used for validation when the object was last stored.

C.1.1.2. Managed Object Schema Validation

Schema validation is performed unequivocally whenever an object is stored, and conditionally whenever an object is retrieved from the data store and exhibits a `_schema_rev` value that differs from the `_rev` of the schema that the OpenIDM instance currently has for that managed object type. Whenever schema validation is performed, the `_schema_rev` of the object is updated to contain the `_rev` value of the current schema.

C.1.1.3. Managed Object Derived Properties

Properties can be defined to be strictly derived from other properties within the object. This allows computed and composite values to be created in the object. Whenever an object undergoes a change, all derived properties are recomputed. The values of derived properties are stored in the data store, and are not recomputed upon retrieval.

C.1.2. Data Consistency

Single-object operations shall be consistent within the scope of the operation performed, limited by capabilities of the underlying data store. Bulk operations shall not have any consistency guarantees. OpenIDM does not expose any transactional semantics in the managed object access API.

All access through the REST API uses the ETag and associated conditional headers: `If-Match`, `If-None-Match`. In operations that modify model objects, conditional headers are mandatory.

C.1.3. Managed Object Triggers

Triggers are user-definable functions that validate or modify object or property state.

C.1.3.1. State Triggers

Managed objects are resource-oriented. A set of triggers is defined to intercept the supported request methods on managed objects. Such triggers are intended to perform authorization, redact, or modify objects before the action is performed. The object being operated on is in scope for each trigger, meaning that the object is retrieved by the data store before the trigger is fired.

If retrieval of the object fails, the failure occurs before any trigger is called. Triggers are executed before any optimistic concurrency mechanisms are invoked. The reason for this is to prevent a potential attacker from getting information about an object (including its presence in the data store) before authorization is applied.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

onCreate

Called upon a request to create a new object. Throwing an exception causes the create to fail.

onRead

Called upon a request to retrieve a whole object or portion of an object. Throwing an exception causes the object to not be included in the result. This method is also called when lists of objects are retrieved via requests to its container object; in this case, only the requested properties are included in the object. Allows for uniform access control for retrieval of objects, regardless of the method in which they were requested.

onUpdate

Called upon a request to store an object. The "old" and "new" objects are in-scope for the trigger. The "old" object represents a complete object as retrieved from the data store. The trigger can elect to change "new" object properties. If as a result of the trigger the object's "old" and "new" values are identical (that is, update is reverted), the update ends prematurely, though successfully. Throwing an exception causes the update to fail.

onDelete

Called upon a request to delete an object. Throwing an exception causes the deletion to fail.

C.1.3.2. Object Storage Triggers

An object-scoped trigger applies to an entire object. Unless otherwise specified, the object itself is in scope for the trigger.

onValidate

Validates an object prior to its storage in the data store. Throws an exception in the event of a validation failure.

onRetrieve

Called when an object is retrieved from the data store. Typically used to transform an object after it has been retrieved (for example decryption, JIT data conversion).

onStore

Called just prior to when an object is stored in the data store. Typically used to transform an object just prior to its storage (for example, encryption).

C.1.3.3. Property Storage Triggers

A property-scoped trigger applies to a specific property within an object. Only the property itself is in scope for the trigger. No other properties in the object should be accessed during execution of the

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

trigger. Unless otherwise specified, the order of execution of property-scoped triggers is intentionally left undefined.

onValidate

Validates a given property value after its retrieval from and prior to its storage in the data store. Throws an exception in the event of a validation failure.

onRetrieve

Called after an object is retrieved from the data store. Typically used to transform a given property after its object's retrieval.

onStore

Called prior to when an object is stored in the data store. Typically used to transform a given property prior to its object's storage.

C.1.3.4. Storage Trigger Sequences

Triggers are executed in the following order:

Object Retrieval Sequence

1. Retrieve the raw object from the data store
2. Call object `onRetrieve` trigger
3. Per-property within the object:
   - Call property `onRetrieve` trigger
   - Perform schema validation if `_schema_rev` does not match (see the *Schema Validation* section)

Object Storage Sequence

1. Per-property within the object:
   - Call property `onValidate` trigger
   - Call object `onValidate` trigger
2. Per-property trigger within the object:
   - Call property `onStore` trigger
   - Call object `onStore` trigger
   - Store the object with any resulting changes to the data store

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

C.1.4. Managed Object Encryption

Sensitive object properties can be encrypted prior to storage, typically through the property `onStore` trigger. The trigger has access to configuration data, which can include arbitrary attributes that you define, such as a symmetric encryption key. Such attributes can be decrypted during retrieval from the data store through the property `onRetrieve` trigger.

C.1.5. Managed Object Configuration

Configuration of managed objects is provided through an array of managed object configuration objects.

```
{
  "objects": [ managed-object-config object, ... ]
}
```

objects

array of managed-object-config objects, required

Specifies the objects that the managed object service manages.

Managed-Object-Config Object Properties

Specifies the configuration of each managed object.

```
{
  "name"      : string,
  "schema"    : json-schema object,
  "onCreate"  : script object,
  "onRead"    : script object,
  "onUpdate"  : script object,
  "onDelete"  : script object,
  "onValidate": script object,
  "onRetrieve": script object,
  "onStore"   : script object,
  "properties": [ property-configuration object, ... ]
}
```

name

string, required

The name of the managed object. Used to identify the managed object in URIs and identifiers.

schema

json-schema object, optional

The schema to use to validate the structure and content of the managed object. The schema-object format is specified by the JSON Schema specification.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

onCreate

script object, optional

A script object to trigger when the creation of an object is being requested. The object to be created is provided in the root scope as an object property. The script may change the object. If an exception is thrown, the create aborts with an exception.

onRead

script object, optional

A script object to trigger when the read of an object is being requested. The object being read is provided in the root scope as an object property. The script may change the object. If an exception is thrown, the read aborts with an exception.

onUpdate

script object, optional

A script object to trigger when an update to an object is requested. The old value of the object being updated is provided in the root scope as an `oldObject` property. The new value of the object being updated is provided in the root scope as a `newObject` property. The script may change the `newObject`. If an exception is thrown, the update aborts with an exception.

onDelete

script object, optional

A script object to trigger when the deletion of an object is being requested. The object being deleted is provided in the root scope as an object property. If an exception is thrown, the deletion aborts with an exception.

onValidate

script object, optional

A script object to trigger when the object requires validation. The object to be validated is provided in the root scope as an object property. If an exception is thrown, the validation fails.

onRetrieve

script object, optional

A script object to trigger once an object is retrieved from the repository. The object that was retrieved is provided in the root scope as an object property. The script may change the object. If an exception is thrown, then object retrieval fails.

onStore

script object, optional

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

A script object to trigger when an object is about to be stored in the repository. The object to be stored is provided in the root scope as an object property. The script may change the object. If an exception is thrown, then object storage fails.

properties

array of property-config objects, optional

A list of property specifications.

Script Object Properties

```
{
  "type"  : "text/javascript",
  "source": string
 }
```

type

string, required

Specifies the type of script to be executed. Currently, only `"text/javascript"` is supported.

source, file

string, required (only one, source or file is required)

Specifies the source code of the script to be executed (if the keyword is "source"), or a pointer to the file that contains the script (if the keyword is "file").

Property Config Properties

```
{
  "name"      : string,
  "onValidate": script object,
  "onRetrieve": script object,
  "onStore"   : script object,
  "encryption": property-encryption object
}
```

name

string, required

The name of the property being configured.

onValidate

script object, optional

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

A script object to trigger when the property requires validation. The property to be validated is provided in the root scope as the `property` property. If an exception is thrown, the validation fails.

onRetrieve

script object, optional

A script object to trigger once a property is retrieved from the repository. The property that was retrieved is provided in the root scope as the `property` property. The script may change the property value. If an exception is thrown, then object retrieval fails.

onStore

script object, optional

A script object to trigger when a property is about to be stored in the repository. The property to be stored is provided in the root scope as the `property` property. The script may change the property value. If an exception is thrown, then object storage fails.

encryption

property-encryption object, optional

Specifies the configuration for encryption of the property in the repository. If omitted or null, the property is not encrypted.

Property Encryption Object

```
{
  "cipher": string,
  "key"   : string
}
```

cipher

string, optional

The cipher transformation used to encrypt the property. If omitted or null, the default cipher of `"AES/CBC/PKCS5Padding"` is used.

key

string, required

The alias of the key in the OpenIDM cryptography service keystore used to encrypt the property.

C.1.6. Custom Managed Objects

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Managed objects in OpenIDM are inherently fully user definable and customizable. Like all OpenIDM objects, managed objects can maintain relationships to each other in the form of links. Managed objects are intended for use as targets and sources for synchronization operations to represent domain objects, and to build up virtual identities. The name comes from the intention that OpenIDM stores and manages these objects, as opposed to system objects that are present in external systems.

OpenIDM can synchronize and map directly between external systems (system objects), without storing intermediate managed objects. Managed objects are appropriate, however, as a way to cache the data—for example, when mapping to multiple target systems, or when decoupling the availability of systems—to more fully report and audit on all object changes during reconciliation, and to build up views that are different from the original source, such transformed and combined or virtual views. Managed objects can also be allowed to act as an authoritative source if no other appropriate source is available.

Other object types exist for other settings that should be available to a script, such as configuration or look-up tables that do not need audit logging.

C.1.6.1. Setting Up a Managed Object Type

To set up a managed object, you declare the object in the `conf/managed.json` file where OpenIDM is installed. The following example adds a simple `foobar` object declaration after the user object type.

```
{
    "objects": [
        {
            "name": "user"
        },
        {
            "name": "foobar"
        }
    ]
}
```

C.1.6.2. Manipulating Managed Objects Declaratively

By mapping an object to another object, either an external system object or another internal managed object, you automatically tie the object life cycle and property settings to the other object. See the chapter on *Configuring Synchronization* for details.

C.1.6.3. Manipulating Managed Objects Programmatically

You can address managed objects as resources using URLs or URIs with the `managed/` prefix. This works whether you address the managed object internally as a script running in OpenIDM or externally through the REST interface.

You can use all resource API functions in script objects for create, read, update, delete operations, and also for arbitrary queries on the object set, but not currently for arbitrary actions. See the *Scripting Reference* appendix for details.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

OpenIDM supports concurrency through a multi version concurrency control (MVCC) mechanism. In other words, each time an object changes, OpenIDM assigns it a new revision.

Objects can be arbitrarily complex as long as they use supported types, such as maps, lists, numbers, strings, and booleans as defined in JSON.

C.1.6.3.1. Creating Objects

The following script example creates an object type.

```
openidm.create("managed/foobar/myidentifier", mymap)
```

C.1.6.3.2. Updating Objects

The following script example updates an object type.

```
var expectedRev = origMap._rev
openidm.update("managed/foobar/myidentifier", expectedRev, mymap)
```

The MVCC mechanism requires that `expectedRev` be set to the expected revision of the object to update. You obtain the revision from the object's `_rev` property. If something else changes the object concurrently, OpenIDM rejects the update, and you must either retry or inspect the concurrent modification.

C.1.6.3.3. Patching Objects

You can partially update a managed object using the patch method, which changes only the specified properties of the object. OpenIDM implements the JSON patch media type version 02, described at https://tools.ietf.org/html/draft-pbryan-json-patch-02.

The following script example updates an object type.

```
openidm.patch("managed/foobar/myidentifier", rev, value)
```

The patch method supports a revision of `"null"`, which effectively disables the MVCC mechanism, that is, changes are applied, regardless of revision. In the REST interface, this matches the `If-Match: "*"` condition supported by patch.

The API supports patch by query, so the caller does not need to know the identifier of the object to change.

```
$ curl
 --header "X-OpenIDM-Username: openidm-admin"
 --header "X-OpenIDM-Password: openidm-admin"
 --request POST -d '[{"replace":"/password","value": "Passw0rd"}]'
 "http://localhost:8080/openidm/managed/user?_action=patch&_queryId=for-userName&uid=DDOE"
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

For the syntax on how to formulate the query `_queryId=for-userName&uid=DDOE` see Section C.1.6.3.6, "Querying Object Sets".

C.1.6.3.4. Deleting Objects

The following script example deletes an object type.

```
var expectedRev = origMap._rev
openidm.delete("managed/foobar/myidentifier", expectedRev)
```

The MVCC mechanism requires that `expectedRev` be set to the expected revision of the object to update. You obtain the revision from the object's `_rev` property. If something else changes the object concurrently, OpenIDM rejects deletion, and you must either retry or inspect the concurrent modification.

C.1.6.3.5. Reading Objects

The following script example reads an object type.

```

val = openidm.read("managed/foobar/myidentifier")
```

C.1.6.3.6. Querying Object Sets

The following script example queries object type instances.

```

var params = {
    "_queryId": "my-custom-query-id",
    "mycustomtoken": "samplevalue"
};
val = openidm.query("managed/foobar", params);
```

The example sets up a query with ID `my-custom-query-id`. The query definition (not shown) is found in the repository configuration. The query definition includes the parameter `mycustomtoken` for token substitution.

An example for a query can be found in chapter *Managed Object as Correlation Query Target* .

C.1.7. Accessing Managed Objects Through the REST API

OpenIDM exposes all managed object functionality through the REST API unless you configure a policy to prevent such access. In addition to the common REST functionality of create, read, update, delete, patch, and query, the REST API also supports patch by query. See the *REST API Reference* appendix for details.

OpenIDM requires authentication to access the REST API. Authentication configuration is shown in `openidm/conf/authentication.json`. The default authorization filter script is `openidm/script/router-authz.js`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

C.2. Configuration Objects

OpenIDM provides an extensible configuration to allow you to leverage regular configuration mechanisms.

Unlike native OpenIDM configuration, which OpenIDM interprets automatically and can start new services, OpenIDM stores custom configuration objects and makes them available to your code through the API.

See the chapter on *Configuration Options* for an introduction to standard configuration objects.

C.2.1. When To Use Custom Configuration Objects

Configuration objects are ideal for metadata and settings that need not be included in the data to reconcile. In other words, use configuration objects for data that does not require audit log, and does not serve directly as a target or source for mappings.

Although you can set and manipulate configuration objects both programmatically and also manually, configuration objects are expected to change slowly, perhaps through a mix of both manual file updates and also programmatic updates. To store temporary values that can change frequently and that you do not expect to be updated by configuration file changes, custom repository objects can be more appropriate.

C.2.2. Custom Configuration Object Naming Conventions

By convention custom configuration objects are added under the reserved context, `config/custom`.

You can choose any name under `config/context`. Be sure, however, to choose a value for *context* that does not clash with future OpenIDM configuration names.

C.2.3. Mapping Configuration Objects To Configuration Files

If you have not disabled the file based view for configuration, you can view and edit all configuration including custom configuration in `openidm/conf/*.json` files. The configuration maps to a file named `context-config-name.json`, where *context* for custom configuration objects is `custom` by convention, and *config-name* is the configuration object name. A configuration object named `escalation` thus maps to a file named `conf/custom-escalation.json`.

OpenIDM detects and automatically picks up changes to the file.

OpenIDM also applies changes made through APIs to the file.

By default, OpenIDM stores configuration objects in the repository. The file view is an added convenience aimed to help you in the development phase of your project.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

C.2.4. Configuration Objects File & REST Payload Formats

By default, OpenIDM maps configuration objects to JSON representations.

OpenIDM represents objects internally in plain, native types like maps, lists, strings, numbers, booleans, null. OpenIDM constrains the object model to simple types so that mapping objects to external representations is trivial.

The following example shows a representation of a configuration object with a look-up map.

```
{
    "CODE123" : "ALERT",
    "CODE889" : "IGNORE"
}
```

In the JSON representation, maps are represented with braces (`{ }`), and lists are represented with brackets (`[ ]`). Objects can be arbitrarily complex, as in the following example.

```
{
    "CODE123" : {
        "email" : ["sample@sample.com", "john.doe@somedomain.com"],
        "sms" : ["555666777"]
    }
    "CODE889" : "IGNORE"
}
```

C.2.5. Accessing Configuration Objects Through the REST API

You can list all available configuration objects, including system and custom configurations, using an HTTP GET on `/openidm/config`.

The `_id` property in the configuration object provides the link to the configuration details with an HTTP GET on `/openidm/config/id-value`. By convention, the *id-value* for a custom configuration object called `escalation` is `custom/escalation`.

OpenIDM supports REST mappings for create, read, update, and delete of configuration objects. Currently OpenIDM does not support patch and custom query operations for configuration objects.

C.2.6. Accessing Configuration Objects Programmatically

You can address configuration objects as resources using the URL or URI `config/` prefix both internally and also through the REST interface. The resource API provides script object functions for create, read, update, and delete operations.

OpenIDM supports concurrency through a multi version concurrency control mechanism. In other words, each time an object changes, OpenIDM assigns it a new revision.

Objects can be arbitrarily complex as long as they use supported types, such as maps, lists, numbers, strings, and booleans.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

C.2.7. Creating Objects

The following script example creates an object type.

```
openidm.create("config/custom/myconfig", mymap)
```

C.2.8. Updating Objects

The following script example updates an object type.

```
var expectedRev = origMap._rev
openidm.update("managed/custom/myconfig", expectedRev, mymap)
```

The MVCC mechanism requires that `expectedRev` be set to the expected revision of the object to update. You obtain the revision from the object's `_rev` property. If something else changes the object concurrently, OpenIDM rejects the update, and you must either retry or inspect the concurrent modification.

C.2.9. Deleting Objects

The following script example deletes an object type.

```
var expectedRev = origMap._rev
openidm.delete("config/custom/myconfig", expectedRev)
```

The MVCC mechanism requires that `expectedRev` be set to the expected revision of the object to update. You obtain the revision from the object's `_rev` property. If something else changes the object concurrently, OpenIDM rejects deletion, and you must either retry or inspect the concurrent modification.

C.2.10. Reading Objects

The following script example reads an object type.

```
val = openidm.read("config/custom/myconfig")
```

C.3. System Objects

System objects are pluggable representations of objects on external systems. They follow the same RESTful resource based design principles as managed objects. There is a default implementation for the OpenICF framework, which allows any connector object to be represented as a system object.

C.4. Audit Objects

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Audit objects house audit data selected for local storage in the OpenIDM repository. For details, see the chapter on *Using Audit Logs*.

C.5. Links

Link objects define relations between source objects and target objects, usually relations between managed objects and system objects. The link relationship is established by provisioning activity that either results in a new account on a target system, or a reconciliation or synchronization scenario that takes a `LINK` action.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Appendix D. Synchronization Reference

The synchronization engine is one of the core services of OpenIDM. You configure the synchronization service through a `mappings` property that specifies mappings between objects that are managed by the synchronization engine.

```
{
  "mappings": [ object-mapping object, ... ]
}
```

D.1. Object-Mapping Objects

An object-mapping object specifies the configuration for a mapping of source objects to target objects.

```
 {
  "name"             : string,
  "source"           : string,
  "target"           : string,
  "links"            : string,
  "validSource"      : script object,
  "validTarget"      : script object,
  "correlationQuery": script object,
  "properties"       : [ property object, ... ],
  "policies"         : [ policy object, ... ],
  "onCreate"         : script object,
  "onUpdate"         : script object,
  "onLink"           : script object,
  "onUnlink"         : script object
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Mapping Object Properties

name

string, required

Uniquely names the object mapping. Used in the link object identifier.

source

string, required

Specifies the path of the source object set. Example: `"managed/user"`.

target

string, required

Specifies the path of the target object set. Example: `"system/ldap/account"`.

links

string, optional

Enables reuse of the links created in another mapping. Example: `"systemLdapAccounts_managedUser"` reuses the links created by a previous mapping whose `name` is `"systemLdapAccounts_managedUser"`.

validSource

script object, optional

A script that determines if a source object is valid to be mapped. The script yields a boolean value: `true` indicates the source object is valid; `false` can be used to defer mapping until some condition is met. In the root scope, the source object is provided in the `"source"` property. If the script is not specified, then all source objects are considered valid.

validTarget

script object, optional

A script used during the target phase of reconciliation that determines if a target object is valid to be mapped. The script yields a boolean value: `true` indicates that the target object is valid; `false` indicates that the target object should not be included in reconciliation. In the root scope, the target object is provided in the `"target"` property. If the script is not specified, then all target objects are considered valid for mapping.

correlationQuery

script object, optional

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

A script that yields a query object to query the target object set when a source object has no linked target. The syntax for writing the query depends on the target system of the correlation. See the section on *Correlation Queries* for examples of some common targets. The source object is provided in the `"source"` property in the script scope.

properties

array of property-mapping objects, optional

Specifies mappings between source object properties and target object properties, with optional transformation scripts.

policies

array of policy objects, optional

Specifies a set of link conditions and associated actions to take in response.

onCreate

script object, optional

A script to execute when a target object is to be created, after property mappings have been applied. In the root scope, the source object is provided in the `"source"` property, projected target object in the `"target"` property and the link situation that led to the create operation in `"situation"`. The `_id` property in the target object can be modified, allowing the mapping to select an identifier; if not set then the identifier is expected to be set by the target object set. If the script throws an exception, then target object creation is aborted.

onUpdate

script object, optional

A script to execute when a target object is to be updated, after property mappings have been applied. In the root scope, the source object is provided in the `"source"` property, projected target object in the `"target"` property, link situation that led to the update operation in `"situation"`. If the script throws an exception, then target object update is aborted.

onLink

script object, optional

A script to execute when a source object is to be linked to a target object, after property mappings have been applied. In the root scope, the source object is provided in the `"source"` property, projected target object in the `"target"` property. If the script throws an exception, then target object linking is aborted.

onUnlink

script object, optional

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

A script to execute when a source and a target object are to be unlinked, after property mappings have been applied. In the root scope, the source object is provided in the `"source"` property, projected target object in the `"target"` property. If the script throws an exception, then target object unlinking is aborted.

result

script object, optional

A script to execute on each mapping event, independent of the nature of the operation. In the root scope, the source object is provided in the `"source"` property, projected target object in the `"target"` property. If the script throws an exception, then target object unlinking is aborted.

The "result" script is executed only during reconciliation operations!

D.1.1. Property Objects

A property object specifies how the value of a target property is determined.

```
{
  "target" : string,
  "source" : string,
  "transform" : script object,
  "condition" : script object,
  "default": value
}
```

Property Object Properties

target

string, required

Specifies the path of the property in the target object to map to.

source

string, optional

Specifies the path of the property in the source object to map from. If not specified, then the target property value is derived from the script or default value.

transform

script object, optional

A script to determine the target property value. The root scope contains the value of the source in the `"source"` property, if specified. If the `"source"` property has a value of `""`, then the entire source object of the mapping is contained in the root scope. The resulting value yielded by the script is stored in the target property.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

condition

script object, optional

A script to determine whether the mapping should be executed or not. The condition has an `"object"` property available in root scope, which (if specified) contains the full source object. For example `"source": "(object.email != null)"`. The script is considered to return a boolean value.

default

any value, optional

Specifies the value to assign to the target property if a non-null value is not established by `"source"` or `"transform"`. If not specified, the default value is `null`.

D.1.2. Policy Objects

A policy object specifies a link condition and the associated actions to take in response.

```
{
  "situation"  : string,
  "action"     : string or script object
  "postAction" : optional, script object
}
```

Policy Object Properties

situation

string, required

Specifies the situation for which an associated action is to be defined.

action

string or script object, required

Specifies the action to perform. If a script is specified, the script is executed and is expected to yield a string containing the action to perform.

postAction

script object, optional

Specifies the action to perform after the previously specified action has completed.

D.1.2.1. Script Object

Script objects take the following form.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
  "type"  : "text/javascript",
  "source": string
}
```

type

string, required

Specifies the type of script to be executed. Currently, OpenIDM supports only `"text/javascript"`.

source

string, required

Specifies the source code of the script to be executed.

D.2. Links

To maintain links between source and target objects in mappings, OpenIDM stores an object set in the repository. The object set identifier follows this scheme.

```
links/mapping
```

Here, *mapping* represents the name of the mapping for which links are managed.

Link entries have the following structure.

```
{
   "_id":string,
   "_rev":string,
   "linkType":string,
   "firstId":string
   "secondId":string,
}
```

_id

string

The identifier of the link object.

_rev

string, required

The value of link object's revision.

linkType

string, required

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The type of the link. Usually then name of the mapping which created the link.

firstId

string, required

The identifier of the first of the two linked objects.

secondId

string

The identifier of the second of the two linked objects.

D.3. Queries

OpenIDM performs the following queries on a link object set.

1. Find link(s) for a given firstId object identifier.

```
SELECT * FROM links WHERE linkType
     = value AND firstId = value
```

   Although a single result makes sense, this query is intended to allow multiple results so that this scenario can be handled as an exception.

2. Select link(s) for a given second object identifier.

```
SELECT * FROM links  WHERE linkType
    = value AND secondId = value
```

   Although a single result makes sense, this query is intended to allow multiple results so that this scenario can be handled as an exception.

D.4. Reconciliation

OpenIDM performs reconciliation on a per-mapping basis. The process of reconciliation for a given mapping includes these stages.

1. Iterate through all objects for the object set specified as `"source"`. For each source object, carry out the following steps.

   a. Look for a link to a target object in the link object set, and perform a correlation query (if defined).

   b. Determine the link condition, as well as whether a target object can be found.

   c. Determine the action to perform based on the policy defined for the condition.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

   d. Perform the action.
   e. Keep track of the target objects for which a condition and action has already been determined.
   f. Write the results.

2. Iterate through all object identifiers for the object set specified as `"target"`. For each identifier, carry out the following steps.
   a. Find the target in the link object set.

      Determine if the target object was handled in the first phase.
   b. Determine the action to perform based on the policy defined for the condition.
   c. Perform the action.
   d. Write the results.

3. Iterate through all link objects, carrying out the following steps.
   a. If the `reconId` is `"my"`, then skip the object.

      If the `reconId` is not recognized, then the source or the target is missing.
   b. Determine the action to perform based on the policy.
   c. Perform the action.
   d. Store the `reconId` identifer in the mapping to indicate that it was processed in this run.

> **Note**
>
> To optimize a reconciliation operation, the reconciliation process does not attempt to correlate source objects to target objects if the set of target objects is empty when the correlation is started. For information on changing this default behaviour, see *Reconciliation Optimization*.

D.5. REST API

External synchronized objects expose an API to request immediate synchronization. This API includes the following requests and responses.

Request

Example:

```
POST /openidm/system/xml/account/jsmith?action=sync HTTP/1.1
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Response (success)

Example:

```
HTTP/1.1 204 No Content
...
```

Response (synchronization failure)

Example:

```
HTTP/1.1 409 Conflict
...
[JSON representation of error]
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Appendix E. REST API Reference

OpenIDM provides a RESTful API for accessing managed objects.

E.1. URI Scheme

The URI scheme for accessing a managed object follows this convention, assuming the OpenIDM web application was deployed at `/openidm`.

```
/openidm/managed/type/id
```

E.2. Object Identifiers

Each managed object has an identifier (expressed as *id* in the URI scheme) which is used to address the object through the REST API. The REST API allows for the client-generated and server-generated identifiers, through PUT and POST methods. The default server-generated identifier type is a UUID. Object identifiers that begin with underscore (`_`) are reserved for future use.

E.3. Content Negotiation

The REST API fully supports negotiation of content representation through the `Accept` HTTP header. Currently, the supported content type is JSON; omitting content-negotiation is equivalent to including the following header:

```
Accept: application/json
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

E.4. Conditional Operations

The REST API fully supports conditional operations through the use of the `ETag`, `If-Match` and `If-None-Match` HTTP headers. The use of HTTP conditional operations is the basis of OpenIDM's optimistic concurrency control system. Clients should make requests conditional in order to prevent inadvertent modification of the wrong version of an object.

E.5. Supported Methods

The managed object API uses standard HTTP methods to access managed objects.

GET

Retrieves a managed object in OpenIDM.

Example Request

```
GET /openidm/managed/user/bdd793f8 HTTP/1.1
...
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 123
ETag: "0"
...

[JSON representation of the managed object]
```

HEAD

Returns metainformation about a managed object in OpenIDM.

Example Request

```
HEAD /openidm/managed/user/bdd793f8 HTTP/1.1
...
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 123
ETag: "0"
```

PUT

Creates or updates a managed object. PUT is the preferred method of creating managed objects.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Example Request: Creating a new object

```
PUT /openidm/managed/user/5752c0fd9509 HTTP/1.1
Content-Type: application/json
Content-Length: 123
If-None-Match: *
...

[JSON representation of the managed object to create]
```

Example Response: Creating a new object

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: 45
ETag: "0"
...

[JSON representation containing metadata (underscore-prefixed) properties]
```

Example Request: Updating an existing object

```
PUT /openidm/managed/user/5752c0fd9509 HTTP/1.1
Content-Type: application/json
Content-Length: 123
If-Match: "0"
...

[JSON representation of managed object to update]
```

Example Response: Updating an existing object (success)

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: 45
ETag: "0"
....
```

This return code may change in a future release.

Example Response: Updating an existing object when no version is supplied (version conflict)

```
HTTP/1.1 409 Conflict
Content-Type: application/json
Content-Length: 89
...

[JSON representation of error]
```

Example Response: Updating an existing object when an invalid version is supplied (version conflict)

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
HTTP/1.1 412 Precondition Required
Content-Type: application/json
Content-Length: 89
...

[JSON representation of error]
```

POST

The POST method allows arbitrary actions to be performed on managed objects. The `_action` query parameter defines the action to be performed.

The `create` action is used to create a managed object. Because POST is neither safe nor idempotent, PUT is the preferred method of creating managed objects, and should be used if the client knows what identifier it wants to assign the object. The response contains the server-generated `_id` of the newly created managed object.

The POST method create optionally accepts an `_id` query parameter to specify the identifier to give the newly created object. If an `_id` is not provided, the server selects its own identifier.

The `patch` action updates a managed object

Example Create Request

```
POST /openidm/managed/user?_action=create HTTP/1.1
Content-Type: application/json
Content-Length: 123
...

[JSON representation of the managed object to create]
```

Example Response

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: 45
ETag: "0"
...

[JSON representation containing metadata (underscore-prefixed) properties]
```

Example Patch Request

```
POST /openidm/managed/user?_action=patch HTTP/1.1
Content-Type: application/json
Content-Length: 123
...

[JSON representation of the managed object to create]
```

Example Response (success, with data)

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=1kke440cyv1vivbrid6ljso7b;Path=/
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json; charset=UTF-8
ETag: "1"
{"_id":"5752c0fd9509","_rev":"2"}
```

DELETE

Deletes a managed object.

Example Request

```
DELETE /openidm/managed/user/c3471805b60f
If-Match: "0"
...
```

Example Response (success)

```
HTTP/1.1 204 No Content
...
```

Deleting an existing object when no version is supplied (version conflict)

```
HTTP/1.1 409 Conflict
Content-Type: application/json
Content-Length: 89
...

[JSON representation of error]
```

Example Response: Deleting an existing object when an invalid version is supplied (version conflict)

```
HTTP/1.1 412 Precondition Required
Content-Type: application/json
Content-Length: 89
...

[JSON representation of error]
```

PATCH

Performs a partial modification of a managed object.

See the JSON Patch Internet-Draft for details.

Example Request

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
PATCH /openidm/managed/user/5752c0fd9509 HTTP/1.1
Content-Type: application/patch+json
Content-Length: 456
If-Match: "0"
...

[JSON representation of patch document to apply]
```

Example Response (success)

```
HTTP/1.1 204 No Content
ETag: "1"
...
```

Updating an existing object when no version is supplied (version conflict)

```
HTTP/1.1 409 Conflict
Content-Type: application/json
Content-Length: 89
...

[JSON representation of error]
```

Example Response: Updating an existing object when an invalid version is supplied (version conflict)

```
HTTP/1.1 412 Precondition Required
Content-Type: application/json
Content-Length: 89
...

[JSON representation of error]
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Appendix F. Scripting Reference

Scripting allows you to customize various aspects of OpenIDM functionality, for example, by providing custom logic between source and target mappings, defining correlation rules, filters, and triggers, and so on.

F.1. Scripting Configuration

You define scripts using script objects, which can either include the code directly in the configuration, or call an external file that contains the script.

```
{
  "type" : "text/javascript",
  "source": string
}
```

or

```
{
  "type" : "text/javascript",
  "file" : file location
}
```

type

string, required

Specifies the type of script to be executed. Currently, OpenIDM supports only`"text/javascript"`.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

source

string, required if file is not specified

Specifies the source code of the script to be executed.

file

string, required if source is not specified

Specifies the file containing the source code of the script to execute.

F.2. Examples

The following example (included in the `sync.json` file) returns `true` if the `employeeType` is equal to`external`, otherwise returns `false`. This script can be useful during reconciliation to establish whether the source object should be a part of the reconciliation, or ignored.

```
"validTarget": {
   "type" : "text/javascript",
   "source": "object.employeeType == 'external'"
}
```

The following example (included in the `sync.json` file) sets the `__PASSWORD__` attribute to `defaultpwd` when OpenIDM creates a target object.

```
"onCreate" : {
  "type" : "text/javascript",
  "source": "target.__PASSWORD__ = 'defaultpwd'"
}
```

The following example (included in the `router.json` file) shows a trigger to create Solaris home directories using a script. The script is located in a file, `/path/to/openidm/script/createUnixHomeDir.js`.

```
{
  "filters" : [ {
    "pattern" : "^system/solaris/account$",
    "methods" : [ "create" ],
    "onResponse" : {
      "type" : "text/javascript",
      "file" : "script/createUnixHomeDir.js"
    }
  } ]
}
```

F.3. Function Reference

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Functions (access to managed objects, system objects, and configuration objects) within OpenIDM are accessible to scripts via the `openidm` object, which is included in the top-level scope provided to each script.

OpenIDM also provides a `logger` object to access SLF4J facilities. The following code shows an example:

```
logger.info("Parameters passed in: {} {} {}", param1, param2, param3);
```

To set the log level, use `org.forgerock.openidm.script.javascript.JavaScript.level` in `openidm/conf/logging.properties`.

F.3.1. openidm.create(id, value)

This function creates a new resource object.

Parameters

id

string

The identifier of the object to be created.

value

object

The value of the object to be created.

Returns

- The created OpenIDM resource object.

Throws

- An exception is thrown if the object could not be created.

F.3.2. openidm.patch(id, rev, value)

This function performs a partial modification of a managed object. Unlike the `update` function, only the modified attributes are provided, not the entire object.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Parameters

id

string

The identifier of the object to be updated.

rev

string

The revision of the object to be updated, or `null` if the object is not subject to revision control.

value

object

The value of the modifications to be applied to the object.

Returns

- The modified OpenIDM resource object.

Throws

- An exception is thrown if the object could not be updated.

F.3.3. openidm.read(id)

This function reads and returns an OpenIDM resource object.

Parameters

id

string

The identifier of the object to be read.

Returns

- The read OpenIDM resource object, or `null` if not found.

F.3.4. openidm.update(id, rev, value)

This function updates a resource object.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Parameters

id

string

The identifier of the resource object to be updated.

rev

string

The revision of the object to be updated, or `null` if the object is not subject to revision control.

value

object

The value of the object to be updated.

Returns

- The modified OpenIDM resource object.

Throws

- An exception is thrown if the object could not be updated.

F.3.5. openidm.delete(id, rev)

This function deletes a resource object.

Parameters

id

string

The identifier of the object to be deleted.

rev

string

The revision of the object to be deleted, or `null` if the object is not subject to revision control.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Returns

- A `null` value if successful.

Throws

- An exception is thrown if the object could not be deleted.

Note that `delete` is a reserved word in JavaScript and this function can therefore not be called in the usual manner. To call delete from a JavaScript, you must specify the call as shown in the following example:

```
openidm['delete']('managed/user/'+ user._id, user._rev)
```

Calling `openidm.delete()` directly from a JavaScript results in an error similar to the following:

```
org.forgerock.openidm.script.ScriptException: missing name after . operator
```

F.3.6. openidm.query(id, params)

This function performs a query on the specified OpenIDM resource object.

Parameters

id

string

The identifier of the object to perform the query on.

params

object

An object containing the query ID and its parameters.

Returns

- The result of the query. A query result includes the following parameters:

  "query-time-ms"

  The time, in milliseconds, that OpenIDM took to process the query.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

"conversion-time-ms"

(For an OrientDB repository only) the time, in milliseconds, taken to convert the data to a JSON object.

"result"

The list of entries retrieved by the query. The result includes the revision (`"_rev"`) of the entry and any other properties that were requested in the query.

The following example shows the result of a custom query that requests the ID, user name, and email address of managed users in the repository. For an OrientDB repository, the query would be something like `select _openidm_id, userName, email from managed_user,`.

```
{
  "conversion-time-ms": 0,
  "result": [
    {
      "email": "bjensen@example.com",
      "userName": "bjensen",
      "_rev": "0",
      "_id": "36bbb745-517f-4695-93d0-998e1e7065cf"
    },
    {
      "email": "scarter@example.com",
      "userName": "scarter",
      "_rev": "0",
      "_id": "cc3bf6f0-949e-4699-9b8e-8c78ce04a287"
    }
  ],
  "query-time-ms": 1
}
```

Throws

- An exception is thrown if the given query could not be processed.

F.3.7. openidm.action(id, params, value)

This function performs an action on the specified OpenIDM resource object.

Parameters

id

string

The identifier of the object on which the action should be performed.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

params

object

An object containing the parameters to pass to the action.

value

object

A value that can be provided to the action for processing.

Returns

- The result of the action. May be `null` if no result is provided.

Throws

- An exception is thrown if the given action could not be executed for any reason.

F.3.8. openidm.encrypt(value, cipher, alias)

This function encrypts a value.

Parameters

value

any

The value to be encrypted.

cipher

string

The cipher with which to encrypt the value, using the form "algorithm/mode/padding" or just "algorithm". Example: `AES/ECB/PKCS5Padding`.

alias

string

The key alias in the key store with which to encrypt the node.

Returns

- The value, encrypted with the specified cipher and key.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Throws

- An exception is thrown if the object could not be encrypted for any reason.

F.3.9. openidm.decrypt(value)

This function decrypts a value.

Parameters

value

any

The value to be decrypted.

Returns

- A deep copy of the value, with any encrypted value decrypted.

Throws

- An exception is thrown if the object could not be decrypted for any reason.

F.3.10. logger.debug(string message, object... params)

Logs a message at DEBUG level.

Parameters

message

string

The message format to log. Params replace `{}` in your message.

params

object

Arguments to include in the message.

Returns

- A `null` value if successful.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Throws

- An exception is thrown if the message could not be logged.

F.3.11. logger.error(string message, object... params)

Logs a message at ERROR level.

Parameters

message

string

The message format to log. Params replace `{}` in your message.

params

object

Arguments to include in the message.

Returns

- A `null` value if successful.

Throws

- An exception is thrown if the message could not be logged.

F.3.12. logger.info(string message, object... params)

Logs a message at INFO level.

Parameters

message

string

The message format to log. Params replace `{}` in your message.

params

object

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Arguments to include in the message.

Returns

- A `null` value if successful.

Throws

- An exception is thrown if the message could not be logged.

F.3.13. logger.trace(string message, object... params)

Logs a message at TRACE level.

Parameters

message

string

The message format to log. Params replace `{}` in your message.

params

object

Arguments to include in the message.

Returns

- A `null` value if successful.

Throws

- An exception is thrown if the message could not be logged.

F.3.14. logger.warn(string message, object... params)

Logs a message at WARN level.

Parameters

message

string

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

The message format to log. Params replace `{}` in your message.

params

object

Arguments to include in the message.

Returns

- A `null` value if successful.

Throws

- An exception is thrown if the message could not be logged.

F.4. Places to Trigger Scripts

Scripts can be triggered at different places, by different events.

In `openidm/conf/sync.json`

Triggered by situation

onCreate, onUpdate, onDelete, onLink, onUnlink

Object filter

vaildSource, validTarget

Correlating objects

correlationQuery

Triggered on any reconciliation

result

Scripts inside properties

condition, transform

In `openidm/conf/managed.json`

onCreate, onRead, onUpdate, onDelete, onValidate, onRetrieve and onStore

Note that `managed.json` supports only one script per hook. If multiple scripts are defined for the same hook, only the last one is kept.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

In `openidm/conf/router.json`

onRequest, onResponse, onFailure

`router.json` supports multiple scripts per hook.

F.5. Variables Available in Scripts

The variables that are available to scripts depend on the triggers that launch the script. The following section outlines the available variables, per trigger.

condition

object

correlationQuery

source

Custom endpoint scripts

request

onCreate

object, source, target

onDelete

object

onLink

source, target

onRead

object

onRetrieve

object, property

onStore

object, property

onUnlink

source, target

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

onUpdate

oldObject, newObject

onValidate

object, property

result

source, target

synchronization situation scripts

`recon.actionParam` - the details of the synchronization operation in progress. This variable can be used for asynchronous callbacks to execute the action at a later stage.

`sourceAction` - a boolean that indicates whether the situation was assessed during the source phase

`source` (if found)

`target` (if found)

The properties from the configured script object.

taskScanner

input, objectID

transform

source

validSource

source

validTarget

target

F.6. Debugging OpenIDM Scripts

OpenIDM includes Eclipse JSDT libraries so you can use Eclipse to debug your OpenIDM scripts during development.

Procedure F.1. To Enable Debugging

Follow these steps to enable debugging using Eclipse.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

1. Install the environment to support JavaScript development in either of the following ways.
   - Download and install Eclipse IDE for JavaScript Web Developers from the Eclipse download page.
   - Add JavaScript Development Tools to your existing Eclipse installation.
2. Create an empty JavaScript project called `External JavaScript Source` in Eclipse.

   Eclipse then uses the `External JavaScript Source` directory in the default workspace location to store sources that it downloads from OpenIDM.
3. Stop OpenIDM.
4. Edit `openidm/conf/boot/boot.properties` to enable debugging.
   a. Uncomment and edit the following line.

   ```
   #openidm.script.javascript.debug=transport=socket,suspend=y,address=9888,trace=true
   ```

   Here `suspend=y` prevents OpenIDM from starting until the remote JavaScript debugger has connected. You might therefore choose to set this to `suspend=n`.

   b. Uncomment and edit the following line.

   ```
   #openidm.script.javascript.sources=/Eclipse/workspace/External JavaScript Source/
   ```

   Adjust `/Eclipse/workspace/External JavaScript Source/` to match the absolute path to this folder including the trailing `/` character. On Windows, also use forward slashes, such as`C:/Eclipse/workspace/External JavaScript Source/`.

   Each time OpenIDM loads a new script, it then creates or overwrites the file in the `External JavaScript Source` directory. Before toggling breakpoints, be sure to refresh the source manually in Eclipse so you have the latest version.
5. Prepare the Eclipse debugger to allow you to set breakpoints.

   In the Eclipse Debug perspective, select the Breakpoints tab, and then click the Add Script Load Breakpoint icon to open the list of scripts.

   In the Add Script Load Breakpoint window, select your scripts, and then click OK.
6. Start OpenIDM, and connect the debugger.

   To create a new debug, configuration click Run > Debug Configurations... > Remote JavaScript > New button, and then set the port to 9888 as shown above.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Appendix G. Router Service Reference

The OpenIDM router service provides the uniform interface to all objects in OpenIDM: managed objects, system objects, configuration objects, and so on.

G.1. Configuration

The router object as shown in `conf/router.json` defines an array of filter objects.

```
{
  "filters": [ filter object, ... ]
}
```

The required filters array defines a list of filters to be processed on each router request. Filters are processed in the order in which they are specified in this array.

G.1.1. Filter Objects

Filter objects are defined as follows.

```
{
  "pattern": string,
  "methods": [ string, ... ],
  "condition": script object,
  "onRequest": script object,
  "onResponse": script object,
  "onFailure": script object
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

"pattern"

string, optional

Specifies a regular expression pattern matching the JSON pointer of the object to trigger scripts. If not specified, all identifiers (including `null`) match.

"methods"

array of strings, optional

One or more methods for which the script(s) should be triggered. Supported methods are: `"create"`, `"read"`, `"update"`, `"delete"`, `"patch"`, `"query"`, `"action"`. If not specified, all methods are matched.

"condition"

script object, optional

Specifies a script that is called first to determine if the script should be triggered. If the condition yields `"true"`, the other script(s) are executed. If no condition is specified, the script(s) are called unconditionally.

"onRequest"

script object, optional

Specifies a script to execute before the request is dispatched to the resource. If the script throws an exception, the method is not performed, and a client error response is provided.

"onResponse"

script object, optional

Specifies a script to execute after the request is successfully dispatched to the resource and a response is returned. Throwing an exception from this script does not undo the method already performed.

"onFailure"

script object, optional

Specifies a script to execute if the request resulted in an exception being thrown. Throwing an exception from this script does not undo the method already performed.

G.1.2. Script Execution Sequence

All "onRequest" and "onResponse" scripts are executed in sequence. First, the "onRequest" scripts are executed from the top down, then the "onResponse" scripts are executed from the bottom up.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
client -> filter 1 onRequest -> filter 2 onRequest -> resource
client <- filter 1 onResponse <- filter 2 onResponse <- resource
```

The following sample `router.json` file shows the order in which the scripts would be executed:

```
{
    "filters" : [
        {
            "onRequest" : {
                "type" : "text/javascript",
                "file" : "script/router-authz.js"
            }
        },
        {
            "pattern" : "^managed/user/.*",
            "methods" : [
                "read"
            ],
            "onRequest" : {
                "type" : "text/javascript",
                "source" : "java.lang.System.out.println('requestFilter 1');"
            }
        },
        {
            "pattern" : "^managed/user/.*",
            "methods" : [
                "read"
            ],
            "onResponse" : {
                "type" : "text/javascript",
                "source" : "java.lang.System.out.println('responseFilter 1');"
            }
        },
        {
            "pattern" : "^managed/user/.*",
            "methods" : [
                "read"
            ],
            "onRequest" : {
                "type" : "text/javascript",
                "source" : "java.lang.System.out.println('requestFilter 2');"
            }
        },
        {
            "pattern" : "^managed/user/.*",
            "methods" : [
                "read"
            ],
            "onResponse" : {
                "type" : "text/javascript",
                "source" : "java.lang.System.out.println('responseFilter 2');"
            }
        }
    ]
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Will produce a log like:

```
requestFilter 1
requestFilter 2
responseFilter 2
responseFilter 1
```

G.1.3. Script Scope

Scripts are provided with the following scope.

```
{
  "openidm": openidm-functions object,
  "request": resource-request object,
  "response": resource-response object,
  "exception": exception object
}
```

"openidm"

openidm-functions object

Provides access to OpenIDM resources.

"request"

resource-request object

The resource-request context, which has one or more parent contexts. Provided in the scope of `"condition"`, `"onRequest"`, `"onResponse"` and `"onException"` scripts.

"response"

openidm-functions object

The response to the resource-request. Only provided in the scope of the `"onResponse"` script.

"exception"

exception object

The exception value that was thrown as a result of processing the request. Only provided in the scope of the `"onException"` script.

An exception object is defined as follows.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
{
  "error": integer,
  "reason": string,
  "message": string,
  "detail": string
}
```

"error"

integer

The numeric code of the exception.

"reason"

string

The short reason phrase of the exception.

"message"

string

A brief message describing the exception.

"detail"

(optional), string

A detailed description of the exception, in structured JSON format, suitable for programmatic evaluation.

G.2. Example

The following example executes a script after a managed user object is created or updated.

```
{
    "filters": [
        {
            "pattern": "^managed/user/.*",
            "methods": [
                "create",
                "update"
            ],
            "onResponse": {
                "type": "text/javascript",
                "file": "scripts/afterUpdateUser.js"
            }
        }
    ]
}
```


Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Appendix H. Embedded Jetty Configuration

OpenIDM 2.1 includes an embedded Jetty web server.

To configure the embedded Jetty server, edit `openidm/conf/jetty.xml`. OpenIDM delegates all connector configuration to `jetty.xml`. OSGi and PAX web specific settings for connector configuration therefore do not have an effect. This lets you take advantage of all Jetty capabilities, as the web server is not configured through an abstraction that might limit some of the options.

The Jetty configuration can reference configuration properties from OpenIDM, such key store details, from OpenIDM's `boot.properties` configuration file.

H.1. Using OpenIDM Configuration Properties in the Jetty Configuration

OpenIDM exposes a `Param` class that you can use in `jetty.xml` to include OpenIDM configuration. The `Param` class exposes Bean properties for common Jetty settings and generic property access for other, arbitrary settings.

H.1.1. Accessing Explicit Bean Properties

To retrieve an explicit Bean property, use the following syntax in `jetty.xml`.

```
<Get class="org.forgerock.openidm.jetty.Param" name="<bean property name>"/>
```

For example, to set a Jetty property for keystore password:

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

```
<Set name="password">
    <Get class="org.forgerock.openidm.jetty.Param" name="keystorePassword"/>
</Set>
```

Also see the bundled `jetty.xml` for further examples.

The following explicit Bean properties are available.

keystoreType

Maps to `openidm.keystore.type`

keystoreProvider

Maps to `openidm.keystore.provider`

keystoreLocation

Maps to `openidm.keystore.location`

keystorePassword

Maps to `openidm.keystore.password`

keystoreKeyPassword

Maps to `openidm.keystore.key.password`, or the key store password if not set

truststoreLocation

Maps to `openidm.truststore.location`, or the key store location if not set

truststorePassword

Maps to `openidm.truststore.password`, or the key store password if not set

H.1.2. Accessing Generic Properties

```
<Call class="org.forgerock.openidm.jetty.Param" name="getProperty">
  <Arg>org.forgerock.openidm.some.sample.property</Arg>
</Call>
```

H.2. Jetty Default Settings

By default the embedded Jetty server uses the following settings.

- An HTTP connector, listening on port 8080

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

- An SSL connector, listening on port 8443
- Same key store/trust store settings as OpenIDM
- Trivial sample realm, `openidm/security/realm.properties` to add users

The default settings are intended for evaluation only. Adjust them according to your production requirements.

H.3. Registering Additional Servlet Filters

You can register generic servlet filters in the embedded Jetty server to perform additional filtering tasks on requests to or responses from OpenIDM. For example, you might want to use a servlet filter to protect access to OpenIDM with an access management product such, as OpenAM. Servlet filters are configured in files named `openidm/conf/servletfilter-name.json`. These servlet filter configuration files define the filter class, required libraries, and other settings.

A sample servlet filter configuration is provided in `openidm/samples/openam`. The sample configuration includes the servlet filter configuration file (`conf/servletfilter-openam.json`) and the extension script that implements the filter (`script/security/populateContext.js`).

The sample servlet filter configuration file is shown below:

```
{
    "classPathURLs" : [
        "file:/jetty_v61_agent/lib/agent.jar",
        "file:/jetty_v61_agent/lib/openssoclientsdk.jar",
        "file:/jetty_v61_agent/lib/",
        "file:/jetty_v61_agent/locale/"
    ],
    "systemProperties" : {
        "openam.agents.bootstrap.dir" : "/jetty_v61_agent/Agent_001/config"
    },
    "requestAttributes" : {
        "openidm.authinvoked" : "servletfilter-openam"
    },
    "scriptExtensions" : {
        "augmentSecurityContext" : {
            "type" : "text/javascript",
            "file" : "script/security/populateContext.js"
        }
    },
    "filterClass" : "com.sun.identity.agents.filter.AmAgentFilter"
}
```

The sample configuration includes the following properties:

`"classPathURLs"`

The URLs to any required classes or libraries that should be added to the classpath used by the servlet filter class

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

`"systemProperties"`

Any additional Java system properties required by the filter

`"requestAttributes"`

The HTTP Servlet request attributes that will be set by OpenIDM when the filter is invoked. OpenIDM expects certain request attributes to be set by any module that protects access to it, so this helps in setting these expected settings.

`"scriptExtensions"`

Optional script extensions to OpenIDM. Currently only `"augmentSecurityContext"` is supported. A script that is defined in `augmentSecurityContext` is executed by OpenIDM after a successful authentication request. The script helps to populate the expected security context in OpenIDM. For example, the login module (servlet filter) might select to supply only the authenticated user name, while the associated roles and user ID can be augmented by the script.

Only JavaScript is supported (`"type":"text/javascript"`). The script can be provided inline (`"source":`*`script source`*) or in a file (`"file":`*`filename`*). The sample filter extends the filter interface with the functionality in the script `script/security/populateContext.js`.

`"filterClass"`

The servlet filter that is being registered

The following additional properties can be configured for the filter:

`"httpContextId"`

The HTTP context under which the filter should be registered. The default is `"openidm"`.

`"servletNames"`

A list of servlet names to which the filter should apply. The default is `"OpenIDM REST"`.

`"urlPatterns"`

A list of URL patterns to which the filter applies. The default is `["/openidm/*", "/openidmui/*"]`.

`"initParams"`

Filter configuration initialization parameters that are passed to the servlet filter `init` method. For more information, see http://docs.oracle.com/javaee/5/api/javax/servlet/FilterConfig.html.

When a servlet filter is used to integrate an access management product, the specific servlet filter that is used, and the configuration that is associated with that filter, is product-specific. The sample configuration in `openidm/samples/openam` is specific to OpenAM.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

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.
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.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

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.

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

Index

A
Architecture, 1
Audit logs, 188
Authentication, 148
    Internal users, 148, 159
    Managed users, 148
    Roles, 150
Authorization, 148, 151

B
Best practices, 154, 199
Business processes, 164

C
Configuration
    Email, 196
    Files, 205
    Objects, 27
    REST API, 29
    Validating, 19
Connectors, 55
    Examples, 65
    Generating configurations, 90
    Object types, 61
    Remote, 56
Correlation queries, 121

D
Data
    accessing, 41

E
Encryption, 157, 159

F
File layout, 205

H
healthcheck, 10

K
Keytool, 18

L
LiveSync, 97
    Scheduling, 126

M
Mappings, 4, 105
    Hooks for scripting, 123
    Scheduled reconciliation, 127

O
Objects
    Audit objects, 227
    Configuration objects, 27
    Links, 228
    Managed objects, 3, 103, 149, 214, 238
        Customizing, 221
        Identifiers, 238
        Passwords, 141
    Object types, 213
    Script access, 41, 245
    System objects, 3, 227
OpenICF, 55

P
Passwords, 141, 159
Policies, 44
Ports
    8080, 212
    8443, 212
    8444, 212
    Disabling, 160

R
Reconciliation, 4, 97
    Scheduling, 126
Resources, 55
REST API, 29, 238
    Listing configuration objects, 29
Roles, 150
Router service, 259

Copyright © 2011-2017 ForgeRock AS. All rights reserved.

S

Schedule
  Examples, 134
Scheduler, 126, 129
  Configuration, 129
Scheduling tasks, 129
Scripting, 244
  Functions, 246
Security, 154
  Authentication, 157
  Encryption, 157, 160
  SSL, 154
Sending mail, 196
Server logs, 54
Starting OpenIDM, 5
Stopping OpenIDM, 5
Synchronization, 4, 97, 229
  Actions, 112
  Conditions, 106
  Connectors, 104
  Correlation queries, 121
  Creating attributes, 106, 111
  Direct (push), 97
  Encryption, 109
  Filtering, 107
  Mappings, 105
  Passwords, 142
    With Active Directory, 144
    With OpenDJ, 143
  Reusing links, 111
  Scheduling, 126
  Situations, 112
  Transforming attributes, 107

T

Troubleshooting, 201

W

Workflow, 164

Copyright © 2011-2017 ForgeRock AS. All rights reserved.