Integrator's Guide

OpenIDM 2.0.3

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

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

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.

Table of Contents

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.");
    }
}
```

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

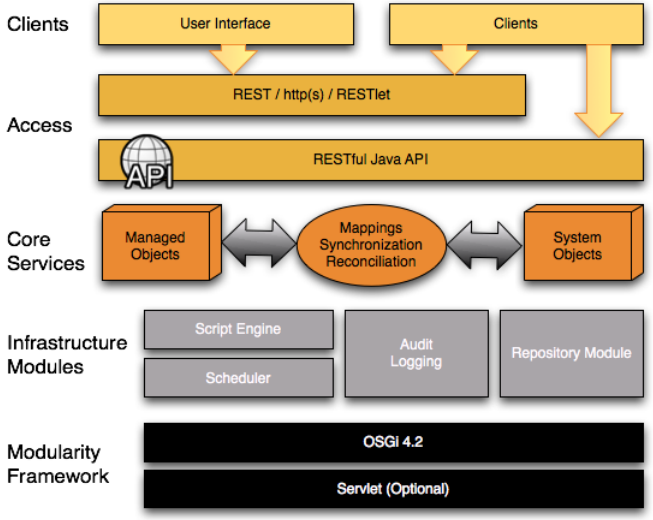
The ForgeRock.org site has links to source code for ForgeRock open source software, as well as links to the ForgeRock forums and technical blogs.

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.

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.

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.

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.

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.

See the *Using Audit Logs* chapter for details.

Repository

The repository provides a common abstraction for a pluggable persistence layer. OpenIDM 2.0.3 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 OpenIDM stores in its pluggable repository. While the default configuration of managed objects is that of a user, any object may be defined through configuration.

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.

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.

Chapter 2
Starting & Stopping OpenIDM

This chapter covers scripts provided for managing OpenIDM.

2.1. Startup & Shutdown

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 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.

```
# 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 &
[3] 454
$ ./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

$
```

   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. Command-Line Tools

OpenIDM includes these command line tools in the `openidm` directory.

cli.sh

This script supports the following subcommands.

validate

Validate all .json configuration files in the `conf/` directory

java -jar bundle/init/openidm-crypto-2.0.3.jar

Utility to obfuscate passwords such as the bootstrap password stored in `openidm/conf/boot/boot.properties`

java -jar bundle/json-crypto-cli-1.1.0.jar

Utility to encrypt and decrypt values in JSON objects

startup.bat
startup.sh

Script to start OpenIDM and the OSGi console

shutdown.sh

Script to stop OpenIDM, especially when you run OpenIDM as a background process

Chapter 3
Configuration Options

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

3.1. About Configuration Objects

OpenIDM exposes internal configuration objects in JSON. Configuration elements can be either have single instances or multiple instances 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 `managed` configuration defines managed objects and their schemas.
- 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 all the mappings OpenIDM uses when synchronizing and reconciling managed objects.

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 instance `provisioner.openicf/xml`.

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

- Multiple `scheduler` configurations can run reconciliations on different schedules.
- Multiple `provisioner.openicf` configurations correspond to the resources connected to OpenIDM.

3.2. When Changing the Configuration

When changing 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 directly editing configuration objects in the internal repository. Use either REST access or JSON files to ensure consistent behavior and that operations are logged.

3.3. 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`.

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

{
    "configurations": [
        {
            "_id": "managed",
            "pid": "managed",
            "factoryPid": null
        },
        {
            "_id": "repo.orientdb",
            "pid": "repo.orientdb",
            "factoryPid": null
        },
        {
            "_id": "scheduler/reconcile_systemXmlAccounts_managedUser",
```

```
            "pid": "scheduler.adc5cd2f-7086-4e30-9d80-b36077861868",
            "factoryPid": "scheduler"
        },
        {
            "_id": "org.apache.felix.fileinstall/openidm",
            "pid":
            "org.apache.felix.fileinstall.abb696a2-95c6-4432-ae74-ba60a319d1bb",
            "factoryPid": "org.apache.felix.fileinstall"
        },
        {
            "_id": "sync",
            "pid": "sync",
            "factoryPid": null
        },
        {
            "_id": "audit",
            "pid": "audit",
            "factoryPid": null
        },
        {
            "_id": "provisioner.openicf/xml",
            "pid": "provisioner.openicf.10e2dd6d-442d-466c-a077-643bb53e2006",
            "factoryPid": "provisioner.openicf"
        },
        {
            "_id": "router",
            "pid": "router",
            "factoryPid": null
        },
        {
            "_id": "authentication",
            "pid": "authentication",
            "factoryPid": null
        }
    ]
}
```


You can find single instance configuration objects 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": "1.1.0.0",
        "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": "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": "array",
                    "items": {
                        "type": "string",
                        "nativeType": "string"
                    },
                    "nativeName": "email",
                    "nativeType": "string"
                },
                "__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": {}
}
```

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

Chapter 4

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 using the following values.

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

Chapter 5

Connecting to External Resources

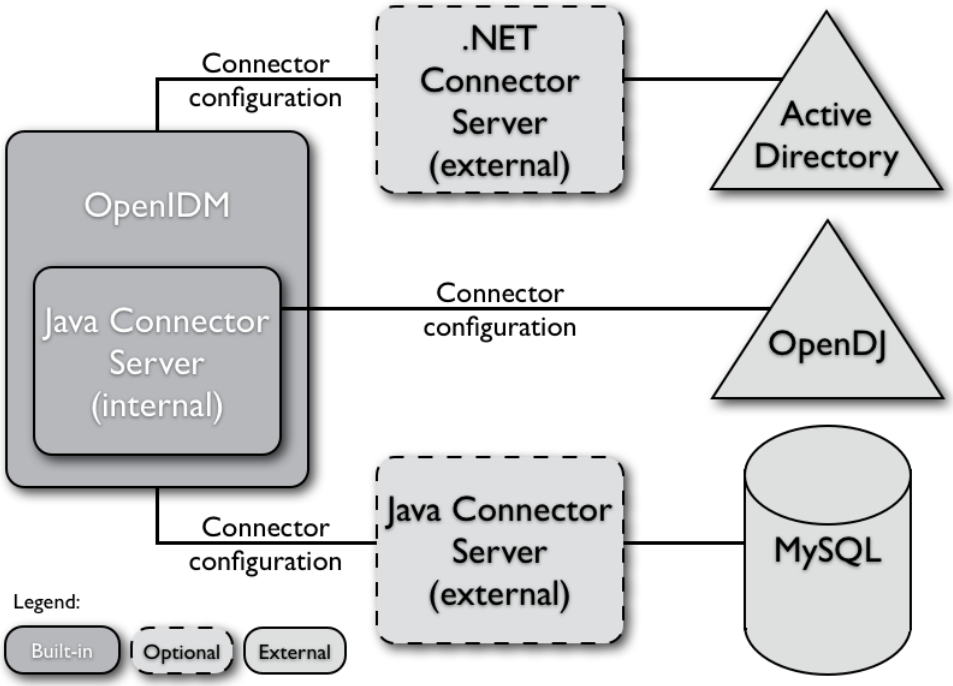
This chapter describes how to connect to external resources such as LDAP, Active Directory, flat files, and others. Configurations shown here are simplified to show essential aspects. Not all resources support all OpenIDM operations, however the resources shown here support most of the CRUD operations,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

5.1. 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.



OpenICF provides a common service provider interface to allow identity services access to the resources containing 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 stand-alone process.

A remote connector server is needed 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 stand alone, 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.

5.2. Accessing Remote Connectors

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

The connector info provider service takes this configuration.

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

Connector Info Provider Properties

connectorsLocation

string, optional

Specifies the directory where 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,
  "useSSL"         : false,
  "timeout"        : 0,
  "key"            : "Passw0rd",
  "trustManagers" :
    [
       "X509TrustManager",
       "BlindTrustManager"
    ]
}
```

OpenIDM supports the following remote connector server object properties.

name

string, required

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

useSSL

boolean, optional

Specifies to use or not 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`.

5.3. Configuring Connectors

Connectors are configured through the OpenICF provisioner service. Each configuration is stored in an extra file/object in the `openidm/conf/` folder or under the same URL respectively. The file name convention is `provisioner.openicf-`*`name`*`.json`.

Note

Though the *name* part of the file name is free text, it must not contain any "-" character!

The following example shows an OpenICF provisioner service configuration.

```
{
 "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"    : "1.1.0.0",
  "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.

- 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.

```
{
  "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" :
```

```
  {
    "$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

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",
               "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

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.

5.4. Connector Configuration Examples

This section explains provisioner configurations for common connectors. Also see Section 5.5, "Creating Default Connector Configurations" for instructions on interactively building connector configurations.

5.4.1. 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": "1.1.0.0",
        "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.

xsdFilePath

References custom schema defining attributes specific to your project

5.4.2. 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.openicf-ldap-connector",
        "bundleVersion": "1.1.0.0"
    }
}
```

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": [],
    "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

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

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.

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

5.4.3. 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.

See the OpenICF Connnector Server page for instructions on installing a .NET connector server. Take care to set the key as described in the instructions.

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 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.

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

5.4.4. 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 now, like all other connectors, in `openidm/connectors`. Therefore the `connectorHostRef` must point to `"#LOCAL"`.

```
{
  "connectorRef": {
    "connectorHostRef": "#LOCAL",
    "connectorName": "org.forgerock.openicf.csvfile.CSVFileConnector",
    "bundleName":
      "org.forgerock.openicf.connectors.file.openicf-csvfile-connector",
    "bundleVersion": "1.1.0.0"
  }
}
```

The following excerpt shows required configuration properties.

```
{
    "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`

5.4.5. 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
- Update

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

5.5. 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
- 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.openicf-ldap-connector",
      "bundleVersion": "1.1.0.0"
    },
    {
      "connectorName": "com.forgerock.openicf.xml.XMLConnector",
      "bundleName":
        "org.forgerock.openicf.connectors.file.openicf-xml-connector",
      "bundleVersion": "1.1.0.0"
    },
    {
      "connectorHostRef":
       "osgi:service/org.forgerock.openicf.framework.api.osgi.ConnectorManager",
      "connectorName": "org.forgerock.openicf.scriptedsql.ScriptedSQLConnector",
      "bundleName":
       "org.forgerock.openicf.connectors.db.openicf-scriptedsql-connector",
      "bundleVersion": "1.1.0.0"
    },
    {
      "connectorHostRef":
       "osgi:service/org.forgerock.openicf.framework.api.osgi.ConnectorManager",
      "connectorName": "org.forgerock.openicf.csvfile.CSVFileConnector",
      "bundleName":
       "org.forgerock.openicf.connectors.file.openicf-csvfile-connector",
      "bundleVersion": "1.1.0.0"
    }
  ]
```

```
}
```

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":"1.1.0.0"}}'
 --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.

```
{
    "connectorRef": {
        "connectorName": "com.forgerock.openicf.xml.XMLConnector",
        "bundleName":
            "org.forgerock.openicf.connectors.file.openicf-xml-connector",
        "bundleVersion": "1.1.0.0"
    },
    "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" : "1.1.0.0"
      },
   "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" :
      {
         "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": "1.1.0.0-EA"
    },
    "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": "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__",
            "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",
                        "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.

Chapter 6
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.

6.1. 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, though 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.

Furthermore, not all resources provide the change log mechanism that LiveSync requires. The change long provides OpenIDM with a list of objects changed since the last request such that

OpenIDM does not need to scan all objects for changes. OpenDJ provides an external change log. Active Directory also provides a change log.

In all cases, OpenIDM relies on mappings that you configure to determine what to synchronize and how to carry out synchronization. LiveSync relies on the set of mappings that you configure once per OpenIDM server, whereas reconciliation also 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 the chapter on *Scheduling Synchronization*. Alternatively, you can start reconciliation through the REST interface.

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

6.2. 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 on which they are 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 with 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.

6.3. Basic Data Flow Configuration

Data flow for synchronization involves three types of configuration files, two of which you typically edit, and also a links table that OpenIDM maintains in its repository, as well as scripts needed to check objects and manipulate attributes. The two types of configuration files you edit are the connector configuration files, with one file per external resource, and the synchronization mappings file, with one file per OpenIDM instance.

Connector configuration files

Connector configuration files are described in the chapter on *Connecting to External Resources*. Connector configuration files are 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 `sn` from 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. Also, the connector file strictly maps external resource objects to OpenIDM objects. To construct attributes and to manipulate their values, you use the synchronization mappings file.

Synchronization mappings file

The synchronization mappings file is `openidm/conf/sync.json`. The synchronization mappings represent 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 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 `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": "(source.email != null)"
                    }
                },
                {
                    "target": "displayName",
```

```
                    "source": "";
                    "transform": {
                        "type": "text/javascript",
                        "source": "(source.lastName +', ' + source.firstName;)"
                    }
                }
            ]
        }
    ]
}
```

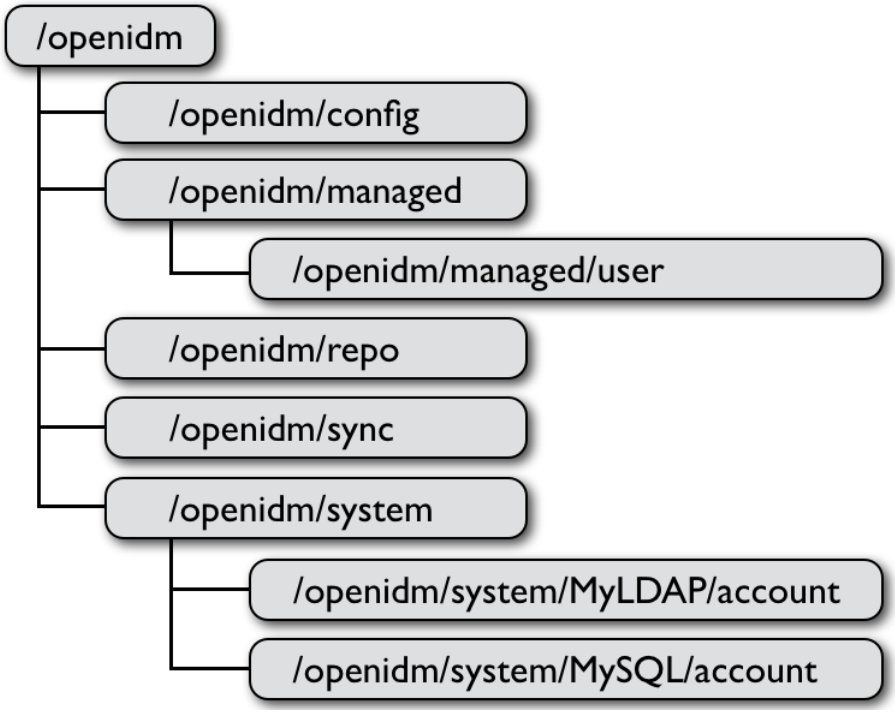
In this example, the source is the external resource, `MyLDAP`, and the target is OpenIDM's repository, specifically the managed user objects. The `properties` reflect OpenIDM attribute names. For example, the mapping has the attribute `lastName` defined in the `MyLDAP` connector configuration file mapped 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`.

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 lets you transform attributes as well. In the example, 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`.



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 matching those defined in the connector configuration for the resource. Many connectors let you limit the scope of objects the connector accesses. For example, the LDAP connector lets you specify base DNs and LDAP filters so you need not access every entry in the directory. Yet, OpenIDM also lets you filter what is considered a 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 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 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.

6.3.1. Using Encrypted Values

OpenIDM supports reversible encryption of attribute values for managed objects. Attribute values to encrypt include passwords (if passwords are not already encrypted on the external resource, which would usually exclude them from the synchronization process, see the chapter about *Passwords*), and also authentication questions, credit card numbers, and social security numbers.

You configure encryption in the managed object configuration. The file to edit is `openidm/conf/managed.json`. 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",
                "file": "script/decryptExtraPassword.js"
            }
        }
    ]
}
```

Do not use the default symmetric key, `openidm-sym-default`, in production. See the chapter on *Securing & Hardening OpenIDM* for instructions on adding your own symmetric key.

6.3.2. Constructing & Manipulating Attributes

OpenIDM lets you construct and manipulate attributes using scripts 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'"
    }
}
```

6.3.3. Reusing Links

When two mappings exist to sync 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.

```
{
    "mappings": [
        {
            "name": "systemMyLDAPAccounts_managedUser",
            "source": "system/MyLDAP/account",
            "target": "managed/user"
        },
        {
            "name": "managedUser_systemMyLDAPAccounts",
            "source": "managed/user",
            "target": "system/MyLDAP/account",
            "links": "systemMyLDAPAccounts_managedUser"
        }
    ]
}
```

6.4. Synchronization Situations & 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"
        },
        {
            "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 sections below.

6.4.1. Synchronization Situations

OpenIDM performs synchronization action in two phases. First, OpenIDM performs the so called source reconciliation, where OpenIDM accounts for source objects and associated links based on the mapping configured. Second, OpenIDM runs the target reconciliation, where OpenIDM iterates over the target objects 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 in the `validSource` property get `qualifies=0`.

2. All records from the appropriate link table

   Objects with corresponding links in the link table of the repository get `link=1`. Objects without corresponding links get `link=0`.

3. All valid objects on the target system

   Object found in the target system get `target=1`. Objects not found in the target system get `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".

"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 is qualified 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. There is one or more targets found through the correlation logic. Detected during change events and reconciliation. Default action: "DELETE".

"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".

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 "validTagrt" script. Only detected during reconciliation. Default action: "IGNORE"

"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. Only detected during reconciliation. Default action: "EXCEPTION".

The following sections reiterate in detail how OpenIDM assigns situations during each of the two synchronization phases.

6.4.2. 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 6.4.1, "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 6.1. Resolving Source Reconciliation Situations

Source Qualifies?		Link Exists?		Target Objects Found[a]			Situation
Yes	**No**	**Yes**	**No**	**0**	**1**	**> 1**	
	X		X	X			SOURCE_IGNORED
	X		X		X		UNQUALIFIED
	X		X			X	UNQUALIFIED
	X	X		X			UNQUALIFIED
	X	X			X		UNQUALIFIED
	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.

6.4.3. 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 6.2. 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

6.4.4. 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.

"IGNORE"

Do not change the link or target object state.

6.5. 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.

6.5.1. 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 ${_resource} WHERE userName = '${uid}'"
```

The following correlation query example shows the JavaScript to call the query defined for OrientDB. The `_query-id` property value matches the name of the query specified in `openidm/conf/repo.orientdb.json`, `for-userName`. The `source.name` value replaces `${uid}` in the query. OpenIDM replaces `${_resource}` in the query with the name of table that holds managed objects.

```
{
  "correlationQuery": {
    "type": "text/javascript",
    "source":
      "var query = {'_query-id' : '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.

6.5.2. 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`.

```
varmap={"query": {"Equals": {"field": "uid", "values": [ source.userName ]}}};
map;
```

6.6. Advanced Data Flow Configuration

Section 6.3, "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",
```

```
                "file": "script/ldapCorrelationQuery.js"
            },
            "properties": [
                {
                    "source": "uid",
                    "transform": {
                        "type": "text/javascript",
                        "source": "source.toLowerCase()"
                    },
                    "target": "userName"
                },
                {
                    "source": "",
                    "transform": {
                        "type": "text/javascript",
                        "source": "if (source.myGivenName)
                            {source.myGivenName;} else {source.givenName;}"
                    },
                    "target": "givenName"
                },
                {
                    "source": "",
                    "transform": {
                        "type": "text/javascript",
                        "source": "if (source.mySn)
                            {source.mySn;} else {source.sn;}"
                    },
                    "target": "familyName"
                },
                {
                    "source": "cn",
                    "target": "fullname"
                },
                {
                    "comment": "Multi-valued in LDAP, single-valued in AD.
                        Retrieve first non-empty value.",
                    "source": "title",
                    "transform": {
                        "type": "text/javascript",
                        "file": "script/getFirstNonEmpty.js"
                    },
                    "target": "title"
                },
                {
                    "condition": {
                        "type": "text/javascript",
                        "source": "var clearObj = openidm.decrypt(object);
                            ((clearObj.password != null) &&
                            (clearObj.ldapPassword != clearObj.password))"
                    },
                    "transform": {
                        "type": "text/javascript",
                        "source": "source.password"
                    },
                    "target": "__PASSWORD__"
                }
            ],
            "onCreate": {
                "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",
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 }
```

Here is the complete list of properties 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/ddoe");
```

6.7. 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 scheduler configuration instead of referencing the `sync.json` mapping.

```
{
    "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": [...]
        }
    }
}
```

Chapter 7
Scheduling Synchronization

OpenIDM provides scheduling for synchronization operations such as LiveSync and reconciliation. You schedule the operations using **cron**-like syntax.

This chapter describes scheduling for reconciliation and LiveSync. Yet, you can use OpenIDM's scheduler service to schedule other events, too. See the *Scheduler Reference* appendix for details.

You configure the scheduler through JSON objects. The corresponding configuration file names take the form `openidm/conf/[org.forgerock.openidm.]scheduler-schedule-name.json`, where `[org.forgerock.openidm.]` is optional and added automatically when OpenIDM reads the configuration, and *schedule-name* is the logical name for the scheduled operation, such as `reconcile_systemXmlAccounts_managedUser`.

7.1. Scheduler Configuration

The scheduler configuration file, `openidm/conf/[org.forgerock.openidm.]scheduler-schedule-name.json` has the following format.

```
{
  "enabled"       : true,
  "type"          : "cron",
  "startTime"     : "(optional) time",
  "endTime"       : "(optional) time",
  "schedule"      : "cron expression",
  "timeZone"      : "(optional) time zone",
  "invokeService" : "service identifier",
  "invokeContext" : "service specific context info"
}
```

- The optional properties `"startTime"`, `"endTime"`, and `"timeZone"` properties can be absent, or empty.

  When specifying times and dates, use ISO 8601 format, `YYYY-MM-DDThh:mm:ss`.

- OpenIDM relies on the Quartz Scheduler. The *cron expression* to use is described in the *CronTrigger Tutorial* and in *Lesson 6: CronTrigger*.

- The `"invokeService"` property takes either `"sync"` for reconciliation or `"provisioner"` for LiveSync.

  You can also specify the service identifier by its full name as in `"invokeService" : "org.forgerock.openidm.sync"`.

- The `"invokeContext"` property depends on the setting for `"invokeService"`.

For LiveSync, `"source"` takes an external resource name and object type.

```
{
    "invokeService": "provisioner",
    "invokeContext": {
        "action": "liveSync",
        "source": "system/ldap/account"
    }
}
```

For reconciliation, `"mapping"` takes the name of the mapping configured in `openidm/conf/sync.json`.

```
{
    "invokeService": "sync",
    "invokeContext": {
        "action": "reconcile",
        "mapping": "systemLdapAccounts_managedUser"
    }
}
```

For reconciliation, you can define the mapping in two ways.

- Referencing the mapping by its name in `sync.json` as shown in the example above. The mapping must exist in the `sync.json` file.
- Configuring the mapping in the scheduler configuration by using the `"mapping"` property as shown in the example in *Alternative Mappings Location*.

7.2. Scheduler 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,
    "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.

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

Chapter 8
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.

8.1. 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 can enforce password policy rules by applying validation rules to attributes of managed user objects. Suppose you want to rule out use of the following user passwords.

- `password`
- `123456`
- `12345678`
- `qwerty`
- `abc123`

You could include the following configuration in `openidm/conf/managed.json` to validate passwords.

```
{
    "objects" : [
        {
            "name" : "user",
            "properties" : [
                {
                    "name" : "password",
                    "encryption" : {
                        "key" : "openidm-sym-default"
                    }
                }
            ],
            "onValidate" : {
                "type" : "text/javascript",
                "file" : "script/password-validator.js"
            }
        }
    ]
}
```

The corresponding script, `openidm/script/password-validator.js`, returns `true` if the password is valid. For example, the following script checks that the password is not one of those listed above.

```
const dictionary = ['password','123456','12345678', 'qwerty', 'abc123'];

function isValidPassword() {
    var cleartextObject = openidm.decrypt(object);
    for (var i = 0; i < dictionary.length; i++) {
        if (cleartextObject.password == dictionary[i]) {
            throw "Password Policy Violation Exception";
        };
    };
};

isValidPassword();
```

To try this script with the default example, update `openidm/conf/managed.json` as shown above, change the sample user's password in `openidm/samples/sample1/data/xmlConnectorData.xml` to something invalid such as `123456`, and add a mapping for the password property to `openidm/conf/sync.json`:

```
"properties" : [
    {
        "source" : "description",
        "target" : "description"
    },
    {
        "source" : "firstname",
        "target" : "givenName"
    },
    {
        "source" : "email",
        "target" : "email"
    },
    {
```

```
        "source" : "lastname",
        "target" : "familyName"
    },
    {
        "source" : "name",
        "target" : "userName"
    },
    {
        "source" : "password",
        "target" : "password"
    },
    {
        "source" : "name",
        "target" : "_id"
    }
],
```

In addition remove `"flags" : ["NOT_READABLE","NOT_RETURNED_BY_DEFAULT"]` from the password property in the `provisioner.openicf-xml.json` file.

The script called for `onValidate` lets you define rules to validate a property's value before OpenIDM stores the object. If the value violates the rules, OpenIDM throws an exception.

The following excerpt from `openidm/logs/openidm0.log.0` shows the exception when trying to reconcile a user having an invalid password.

```
Jan 16, 2012 10:21:11 AM
 org.forgerock.openidm.sync.impl.ObjectMapping createTargetObject
WARNING: Failed to create target object
org.forgerock.openidm.objset.ForbiddenException:
 Password Policy Violation Exception
```

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.

You can configure OpenIDM to control password values as they are provisioned as shown in the previous examples.

To change the default administrative user password, `openidm-admin`, see the procedure, *To Replace the Default User & Password*, for instructions.

Password recovery

Password recovery involves recovering a password or setting a new password when the password has been forgotten.

OpenIDM can provide a self-service end user interface for password changes, password recovery, and password reset.

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.

8.2. 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 are stored in encrypted form. The plugins then send intercepted password values to OpenIDM over an encrypted channel.

Procedure 8.1. 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 1.0.0, from the OpenIDM download page.

3. Unzip the module delivery.

```
$ unzip ~/Downloads/opendj-accountchange-handler-1.0.0-SNAPSHOT.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.

```
$ ./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 8.2. To Install the Active Directory Password Synchronization Plugin

You 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, OpenIDM Agents - AD 2.0.1, from the OpenIDM download page.

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 ${_resource} WHERE userName = '${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.

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 8.3. 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.

   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 on the directory server and run reconciliation, you notice 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==...
```

Chapter 9

Managing Authentication, Authorization & RBAC

OpenIDM currently provides a simple, yet flexible authentication and authorization mechanism based on REST interface URLs and on roles stored in the repository.

9.1. OpenIDM Users

OpenIDM distinguishes between internal users and managed users.

9.1.1. Internal Users

OpenIDM sets up two internal users by default, anonymous and openidm-admin. OpenIDM separates these accounts from all other accounts to protect them from any reconciliation or sync processes. You can add or remove internal users at any time.

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, `amonymous`, 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 fall back in case other users are locked out. Do not use `openidm-admin` for normal tasks. Usually no one is associated with the `openidm-admin` user, so audit log records pertaining to `openidm-admin` do not reflect the actions of any real person.

OpenIDM encrypts the password, `openidm-admin`, by dfault. Change the password immediately after installation. For instructions, see *To Replace the Default User & Password*.

OpenIDM stores internal users and their role membership in a table in the repository called `internaluser` when implemented in MySQL.

9.1.2. Managed Users

External users that OpenIDM manages are referred to as managed users. OpenIDM stores managed users in the managed objects table of the repository, called `managedobjects` when implemented in MySQL. A second table, `managedobjectproperties` in MySQL, serves as the index table.

By default, the attribute names for managed user login and password are `email` and `password`, respectively.

9.2. 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 9.1.1, "Internal Users". In production, only applications are expected to access the REST interface.

OpenIDM supports an improved authentication mechanism on the REST interface. The reason for not using standards, like basic authentication or form based authentication, is their leak of compatibility with AJAX.

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 details about the OpenIDM authentication mechanism please see *Use Message Level Security* .

You can change the attributes OpenIDM uses to store user login and password values. The attribute names are shown in a database query defined in `openidm/conf/repo.repo-type.json`.

Two queries are defined by default.

`credential-internaluser-query`

Uses the user object ID for login

`credential-query`

Uses the user email attribute for login

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": [
        "openidm-authorized"
    ]
}
```

9.3. 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 default roles assigned to successfully authenticated users authentication using the `defaultUserRoles` property in `openidm/conf/authentication.json`, which takes a list. The default value is `openidm-authorized`.

```
{
    "queryId": "credential-query",
    "queryOnResource": "managed/user",
    "defaultUserRoles": [
        "openidm-authorized"
    ]
}
```

9.4. Authorization

OpenIDM access control can be based on REST interface URLs.

The `openidm/script/router-authz.js` script controls access to REST interface URLs. OpenIDM calls the script for each request. The script either throws the string `Access denied`, or nothing. If it throws `Access denied`, then OpenIDM denies the request. The default script is shown below.

```
const allowCert = false;

function contains(a, o) {
    if (typeof(a) != 'undefined' && a != null) {
        for (var i = 0; i <= a.length; i++) {
            if (a[i] === o) {
                return true;
            }
        }
    }
    return false;
}

function allow() {
    if (typeof(request.parent) === 'undefined' ||
            request.parent.type != 'http') {
        return true;
    }
    var roles = request.parent.security['openidm-roles'];
    if (contains(roles, 'openidm-admin')) {
        return true;
    } else if (allowCert && contains(roles, 'openidm-cert')) {
        return true;
    } else {
        return false;
    }
}

if (!allow()) {
    throw "Access denied";
}
```

The script can be seen as having three parts: constants, functions, and a decision.

constants

The constants can function as global switches, for example to toggle whether certificate-based authentication is allowed.

The example that follows shows a constant used to toggle whether anonymous authentication is allowed.

functions

The default script defines two functions. The `allow()` function is called by the final if-statement. The `contains()` function takes the list of roles assigned to an authenticated user, and checks whether the list contains the role specified as the second argument. If the list does contain the role, then `contains()` returns `true`.

decision

The if-statement at the end of the script executes first. It calls the `allow()` function, which calls other functions. The `allow()` function returns true or false, triggering the behavior of the final if-statement.

The following example extends the script to filter out requests not under `/openidm/config` or `/openidm/managed`.

```
const allowCert = false;
const allowAnon = false;

function contains(a, o) {
    if (typeof(a) != 'undefined' && a != null) {
        for (var i = 0; i <= a.length; i++) {
            if (a[i] === o) {
                return true;
            }
        }
    }
    return false;
}

function matchesContext(path, context, allowRoot, allowSubcontext) {
    if (allowRoot && path === context) {
        return true;
    }
    if (allowSubcontext && path.substring(0, context.length + 1)
            === context + "/") {
        return true;
    }
    return false;
}

function allow() {
    if (typeof(request.parent) === 'undefined' ||
            request.parent.type != 'http') {
        return true;
    }

    // Restrict the URLs that are accessible externally
    var path = request.parent.path;
    if (!(matchesContext(path, "/openidm/config", true, true)
            || matchesContext(path, "/openidm/managed", false, true))) {
        return false;
    }

    var roles = request.parent.security['openidm-roles'];
    if (contains(roles, 'openidm-admin')) {
        return true;
    } else if (allowCert && contains(roles, 'openidm-cert')) {
        return true;
    } else if (allowAnon && contains(roles, 'openidm-reg')) {
        return true;
    } else {
        return false;
    }
}

if (!allow()) {
    throw "Access denied";
}
```

Compared to the default, this example has a second global constant, `allowAnon`, used to allow or deny anonymous access. The new constant serves in the `allow()` function when checking for `openidm-reg` membership, the role for anonymous users.

Furthermore this example includes an additional function, `matchesContext()`, called from the `allow()` function before the role test. The additional test filters out all requests not to `/openidm/config` or below `/openidm/managed`. The context root itself is excluded in the latter call to avoid actions like patch by query on `/openidm/managed/user`.

Chapter 10
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.

10.1. 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 protect with message level security where appropriate.

10.2. 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 it puts into 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.

10.3. 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 a 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.

The following examples show successful authentications.

```
$ curl
 --dump-header /dev/stdout
 --user openidm-admin:openidm-admin
 "http://localhost:8080/openidm/managed/user/?_query-id=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/?_query-id=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"
 "http://localhost:8080/openidm/managed/user/?_query-id=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 penultimate request. You can also request one-time authentication without a session.

```
$ 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/?_query-id=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-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.

10.4. Replace Default Security Settings

Default security settings facilitate evaluation. Change the default encryption key, and then replace the default user password

Procedure 10.1. To Change Default Encryption Keys

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 10.2. To Replace the Default User & Password

After changing the default encryption key, change at least the default user password.

1. Get the encrypted version of the new password.

```
$ cd /path/to/openidm/
$ echo \"newpwd\" > /tmp/newpwd.json
$ java -jar bundle/json-crypto-cli-1.1.0.jar
 -encrypt
 -keystore security/keystore.jceks
 -storetype jceks
 -storepass changeit
 -alias "new-sym-key"
 -srcjson /tmp/newpwd.json

{
    "$crypto": {
        "value": {
            "iv": "sL0KM93PmvvvaQQ8rP+QAQ==",
            "data": "eSH8YgeezoWsDDIvX1WQ2A==",
            "cipher": "AES/CBC/PKCS5Padding",
            "key": "new-sym-key"
        },
        "type": "x-simple-encryption"
    }
}
```

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.

10.5. Secure Jetty

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>
-->
```

10.6. 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 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.

See the chapter on *Managing Authentication, Authorization & RBAC* for an example showing how to protect sensitive URLs.

10.7. 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.

10.8. 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, invoke `openidm-crypto-2.0.3.jar` from the `openidm/` directory.

```
$ cd /path/to/openidm
$ java -jar bundle/init/openidm-crypto-2.0.3.jar
```

This utility helps obfuscate passwords to prevent casual observation. It is not securely encrypted and needs further measures to prevent disclosure. Enter the password as shown.

```
OBF:1v2j1uum1xtv1zej1zer1xtn1uvk1v1v
CRYPT:1206319abab995251d745b151b73131c
```

10.9. 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
```

10.10. 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.

10.11. 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.

10.12. 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 details see the section titled *Startup & Shutdown*.

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.

Chapter 11

Integrating Business Processes & Workflow

Key to any identity management solution is the ability to provide workflow driven provisioning activities, whether for self-service actions such as request 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.

Note

The embedded workflow and business process engine is currently provided as part of a separate, experimental download, the OpenIDM 2.0.3 Experimental Nightly build available from the download page.

You install the experimental download in the same way as standard OpenIDM. See the *Installation Guide* in the *Installation Guide* for instructions. After starting OpenIDM, run the **scr list** command at the console, and check that the bundle is active.

```
-> scr list
...
[  14] [active       ] org.forgerock.openidm.workflow.activiti
...
```

Contact ForgeRock at info@forgerock.com for details on support for the experimental embedded workflow and business process engine.

More information about Activiti and the Activiti project can be found at http://www.activiti.org.

11.1. About BPMN 2.0 & Activity 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 also can be executed as is on any BPMN 2.0-compliant engine, such as the engine embedded in OpenIDM.

Using BMPN 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 such as **vi**, 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.

11.2. Known Issues & Limitations

The following are known issues and limitation for the experimental embedded workflow and business process engine.

- OpenIDM does not include a form generator, making it difficult to include embedded forms (OPENIDM-468).
- Error handling needs improvement.
- This version depends on Activiti 5.8, which does not have the fix for ACT-583: Processes are not found in the .bar file if they are below root.

  To work around this issue, create a directory inside the .bar, and put the BPMN XML artifact in the directory so that it is properly picked up by the Process Engine.

11.3. Invoking Activiti Workflows

You can invoke workflows and business processes from any trigger point OpenIDM offers, including reacting to situations discovered during reconciliation. Invocation relies on the `openidm.action()` function.

```
/*
 * Calling 'myWorkflow' workflow
 */
var map = {
 "_action" : "myWorkflow",
 "_workflowParams" : {
    "foo" : "bar"
  }
};

openidm.action("workflow/activiti", map);
```

11.4. Example Activiti Workflows With OpenIDM

This section describes two example workflows, one using email notification, the other involving a sunset process triggered during reconciliation.

11.4.1. 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"
 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="no-reply@forgerock.com"
      ></activiti:field>
      <activiti:field name="subject" expression="Simple Email Notification"
      ></activiti:field>
      <activiti:field name="html">
        <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.

Before you deploy the .bar, work around the issue mentioned above by adding the XML artifact in the right place in the .bar, for example by expanding the .bar, moving the `EmailNotification.bpmn20.xml` file inside a directory, and recreating the .bar.

After you deploy the .bar, create a script in `openidm/script/invokeEmailNotification.js`. The script invokes the workflow.

```
/*
 * Calling 'EmailNotification' workflow
 */
var map = {
 "_action" : "emailNotification",
 "_workflowParams" : {
    "fromSender" : "noreply@openidm",
    "toEmail" : "john.doe@corp.com"
  }
};

openidm.action("workflow/activiti", map);
```

Also, create a schedule configuration object in `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/invokeEmailNotification.js"
       },
   }
}
```

11.4.2. Example Sunset Workflow Triggered By Reconciliation

OpenIDM can allow deferred deprovisioning and disabling, such as needed when an external consultant's contract expires or is up for renewal. This example shows how OpenIDM can be used to handle such scenarios.

To understand the scenario, consider the target resource and the authoritative source.

Set up a CSV file connector to connect to an authoritative source file containing the following.

```
firstName,uid,"lastName","email","employeeNumber",password,"sunrise","sunset","active"
"Darth","DDOE","Doe","doe@...","123456","Z29vZA==","2011-11-30T...","2011-12-23T...","TRUE"
```

The authoritative source contains the field, `sunset`, containing a time and date string referring to a time in the future.

Set up a simple XML connector to replicate fields in the authoritative CSV source on a 1-to-1 basis.

Define a Sunset Workflow using Activiti Eclipse BPMN 2.0 Designer.

Use the following XML artifact.

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions id="definitions"
            xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
            xmlns:activiti="http://activiti.org/bpmn"
            targetNamespace="Examples"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL
              http://www.omg.org/spec/BPMN/2.0/20100501/BPMN20.xsd">

    <process id="sunset" name="sunset usecase">
       <startEvent id="start" />
       <sequenceFlow sourceRef="start" targetRef="saveInvokeContext"/>
       <scriptTask id="saveInvokeContext" scriptFormat="groovy"
         activiti:returnVariable="invokecontext">
           <script>
               <![CDATA[
               invokecontext =
                       org.forgerock.openidm.context.InvokeContext.getContext()
              ]]>
           </script>
       </scriptTask>
       <sequenceFlow sourceRef="saveInvokeContext"
         targetRef="timerintermediatecatchevent1"/>

       <intermediateCatchEvent id="timerintermediatecatchevent1"
         name="TimerCatchEvent">
           <extensionElements>
               <activiti:field name="t" expression="${time}"
               ></activiti:field>
           </extensionElements>
           <timerEventDefinition>
               <timeDate >${time}</timeDate>
           </timerEventDefinition>
       </intermediateCatchEvent>

       <sequenceFlow sourceRef="timerintermediatecatchevent1"
         targetRef="idmCall"/>


       <serviceTask id="idmCall" activiti:delegateExpression="${openidm}"
         activiti:async="true">
           <extensionElements>
               <activiti:field name="userName" expression="${userName}"
               ></activiti:field>
               <activiti:field name="system" expression="${system}"
               ></activiti:field>
           </extensionElements>
       </serviceTask>
```

```
      <sequenceFlow sourceRef="idmCall" targetRef="readUserAccount"/>
      <scriptTask id="readUserAccount" scriptFormat="groovy"
        activiti:returnVariable="user" >
          <script>
              <![CDATA[
              out:println '********Sunset date for user ' + userName
              org.forgerock.openidm.context.InvokeContext.getContext().
                      putApprover(invokecontext.getApprover())
              org.forgerock.openidm.context.InvokeContext.getContext().
                      putRequester(invokecontext.getRequester())
              org.forgerock.openidm.context.InvokeContext.getContext().
                      pushActivityId(invokecontext.popActivityId())

              user = openidm.read(system + userName);
              ]]>
          </script>
      </scriptTask>
      <sequenceFlow sourceRef="readUserAccount" targetRef="sendMail"/>

      <serviceTask id="sendMail" name="Email Notification"
        activiti:type="mail">
          <extensionElements>
              <activiti:field name="to" expression="${toEmail}"
              ></activiti:field>
              <activiti:field name="from" expression="no-reply@forgerock.com"
              ></activiti:field>
              <activiti:field name="subject" expression="Disabling User"
              ></activiti:field>
              <activiti:field name="text">
                  <activiti:expression><![CDATA[
                  The following user has been disabled:
                  ${user}]]>
                  </activiti:expression>
              </activiti:field>
          </extensionElements>
      </serviceTask>

      <sequenceFlow sourceRef="sendMail" targetRef="disableAccount"/>
      <scriptTask id="disableAccount" scriptFormat="groovy">
          <script>
              <![CDATA[
              out:println '********Disabling user account ' + userName
              user['__ENABLE__'] = false
              openidm.update(system + userName, null, user)
              ]]>
          </script>
      </scriptTask>
      <sequenceFlow sourceRef="disableAccount" targetRef="end"/>
      <endEvent id="end"/>

    </process>
</definitions>
```

Set up a deferred action during reconciliation by reacting to a `FOUND` situation. When called, the script invoked must return the proper action to the found situation, which in the case of this example is `LINK`.

Add the following script to `script/triggerSunset.js` where you installed OpenIDM, invoked when reconciliation encounters a `FOUND` situation.

```
/*
 * Calling 'sunrise' workflow
 */
var map = {
 "_action" : "sunrise",
 "_workflowParams" : {
    "userName" : target.__UID__,
    "system" : "system/xmlfile/account/",
    "time" : source.sunset,
    "toEmail" : "manager@corp.org",
    "fromSender" : "noreply@openidm"
 }
};

openidm.action("workflow/activiti", map);

"LINK"
```

Add the following schedule to `conf/scheduler-reconcile_HR_XML.json` where you installed OpenIDM to invoke the script.

```
{
    "enabled" : true,
    "type" : "cron",
    "schedule" : "0 08 16 * * ?",
    "invokeService" : "sync",
    "invokeContext" : {
       "action" : "reconcile",
       "mapping" : {
           "name" : "CSV_XML",
           "source" : "system/CSV/persons",
           "target" : "system/xmlfile/account",
           "correlationQuery" : {
               "type" : "text/javascript",
               "source" : "var myarray = [source.uid];var map = {
                   'query' : {
                       'Equals': {'field' : 'name','values' : myarray}
                   }
               };map;"
           },
           "properties" : [
               {
                   "source" : "firstname",
                   "target" : "firstname"
               },
               {
                   "source" : "uid",
                   "target" : "name"
               },
               {
                   "source" : "lastname",
                   "target" : "lastname"
               },
               {
```

```
                    "source" : "email",
                    "target" : "email"
                }
            ],
            "policies" : [
                {
                    "situation" : "CONFIRMED",
                    "action" : "IGNORE"
                },
                {
                    "situation" : "FOUND",
                    "action" : {
                        "type" : "text/javascript",
                        "file" : "script/triggerSunset.js"
                    }
                },
                {
                    "situation" : "ABSENT",
                    "action" : "IGNORE"
                },
                {
                    "situation" : "AMBIGUOUS",
                    "action" : "IGNORE"
                },
                {
                    "situation" : "MISSING",
                    "action" : "IGNORE"
                },
                {
                    "situation" : "UNQUALIFIED",
                    "action" : "IGNORE"
                },
                {
                    "situation" : "UNASSIGNED",
                    "action" : "IGNORE"
                }
            ]
        }
    }
}
```

Chapter 12
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, to access details, to 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.

12.1. 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 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.

12.2. 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, such as `"2012-01-10T10:45:42"`

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.

`"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, such as `"2012-01-10T10:45:42"`

Reconciliation Log Fields

Reconciliation messages are split into the following fields.

`"_id"`

UUID for the message object, such as `"0419d364-1b3d-4e4f-b769-555c3ca098b0"`

`"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 sychronization 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, such as `"2012-01-10T10:45:42"`.

12.3. Audit Configuration

OpenIDM exposes the audit logging configuration under `http://host:port/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"
                ]
            }
        },
        "recon": {}
    },
    "logTo": [
        {
            "logType": "csv",
            "location": "audit",
            "recordDelimiter": ";"
        },
        {
            "logType": "repository"
        }
    ]
}
```

12.3.1. 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 lets you configure under what conditions OpenIDM writes to the activity log.

`read`

When an object is read by using its identifier

`create`

When an object is created.

`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.

The `logTo` list lets you define where OpenIDM writes the log.

`csv`

Write to a comma-separated variable format file in the location specified relative to the directory where you installed OpenIDM.

Audit file names are fixed, `access.csv`, `activity.csv`, and `recon.csv`.

`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_activity`, `audit_activity`, and `audit_recon` tables.

In a JDBC repository, OpenIDM stores records in `auditaccess`, `auditactivity`, and `auditrecon` tables.

12.4. 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.

```
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)
```

Chapter 13
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 13.1. 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

`"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
...
```

13.1. 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 `/openam/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"
```

13.2. 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"`.

Chapter 14

OpenIDM Project Best Practices

This chapter lists points to check when implementing an identity management solution with OpenIDM.

14.1. 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.

14.1.1. 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.

14.1.2. 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

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.

14.1.3. 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.

14.1.4. Build

This phase builds and tests the solution prior to moving the solution into production.

14.1.5. Production

This phase deploys the solution into production until an application steady state is reached and maintenance routines and procedures can be applied.

Chapter 15

Troubleshooting

When things are not working check this chapter for tips and answers.

15.1. 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/`.

15.2. 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"
}
```

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.

15.3. 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
```

15.4. 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"
```

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.

15.5. 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.

15.6. 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",`.

- If the connector .jar is installed as a connector under `openidm/connectors`, then the value must be `"connectorHostRef" : "#LOCAL",`.

15.7. 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.

Appendix A. File Layout

When you unpack and start OpenIDM 2.0.3, you create the following files and directories.

`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

`openidm/bundle/`

OSGi bundles and modules required by OpenIDM. Upgrade can install new and upgraded bundles here.

`openidm/bundle/json-crypto-cli-1.1.0.jar`

Utility to encrypt and decrypt values in JSON objects

`openidm/cli.sh`

Management commands for operations such as validating configuration files

`openidm/conf/`

OpenIDM configuration files, including .properties files and JSON views. You can also access JSON views through the REST interface.

`openidm/conf/audit.json`

Audit event publisher configuration view

`openidm/conf/authentication.json`

Authentication configuration view for access to the REST API

`openidm/conf/boot/boot.properties`

OpenIDM bootstrap properties

`openidm/conf/config.properties`

Felix and OSGi bundle configuration properties

`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 view

`openidm/conf/provisioner.openicf-xml.json`

Sample XML connector configuration view. After evaluation, replace this with your own connector configurations.

Each connector instance has a corresponding `provisioner.openicf-`*`name`*`.json` configuration file. Each file specifies connector configuration details such as network information, credentials, attribute schema, and which OpenICF features are supported.

`openidm/conf/repo.orientdb.json`

OrientDB internal repository configuration view

`openidm/conf/router.json`

Router service configuration view

`openidm/conf/scheduler-reconcile_systemXmlAccounts_managedUser.json`

Sample XML scheduler configuration view. After evaluation, replace this with your own configurations.

`openidm/conf/sync.json`

Sample XML synchronization configuration view. After evaluation, replace this with your configuration to describe all mappings used by OpenIDM for synchronization and reconciliation.

`openidm/conf/system.properties`

System configuration properties used when starting OpenIDM services

`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/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/sample/misc/`

Sample configuration files

`openidm/sample/provisioners/`

Sample connector configuration files

`openidm/sample/sample1/`

XML file connector sample installed with OpenIDM

`openidm/sample/sample2/`

OpenDJ connector sample with no back link

`openidm/sample/sample2b/`

OpenDJ connector sample with back link

`openidm/sample/sample3/`

Scripted SQL connector sample for MySQL

`openidm/sample/sample4/`

CSV connector sample

`openidm/sample/sample5/`

LDAP to OpenIDM to Active Directory attribute flow sample using XML resources rather than actual directories

`openidm/sample/sample6/`

LiveSync sample for use with one or two LDAP servers

`openidm/sample/schedulers/`

Sample scheduler configuration files

`openidm/sample/security/`

Sample key store, trust store, and certificates

`openidm/script/`

OpenIDM location for JavaScript files referenced in the configuration

`openidm/script/router-authz.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

`openidm/workflow/`

OpenIDM location for BPMN 2.0 workflows and .bar files

Appendix B. Ports Used

By default the OpenIDM 2.0.3 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.

Appendix C. Data Models & Objects Reference

OpenIDM allows you to customize of 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 when it is appropriate to manage 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.	
Audit objects	Houses audit data in the OpenIDM internal repository.	
Links	Defines a relation between two objects.	

C.1. Accessing Objects

OpenIDM's uniform programming model means that all objects are queried and manipulated in the same way from your scripts using the Resource API. The URL or URI used to identify the target object for an operation depends on the object type. Also additional functionality is available for different types of object, known as *Resource Sets*.

For example, you get managed objects, configuration objects, and repository objects in the following way using the Resource API. See the section on *URI Scheme* for information on how to construct and object ID.

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

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

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

You update entire objects with the `update()` function.

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

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

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

The `create()`, `delete()`, and `query()` functions work in similar fashion.

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

By default, the HTTP GET returns a JSON representation of the object. See the *REST API Reference* appendix for details.

C.2. 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.2.1. 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.2.1.1. 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.

_schema_rev

string

The revision of the schema that was used for validation when the object was last stored.

C.2.1.2. 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 a schema validation is performed, the `_schema_rev` of the object is updated to contain the `_rev` value of the current schema.

C.2.1.3. 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 value of derived properties are stored in the data store, and are not recomputed upon retrieval.

C.2.2. 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.2.3. Managed Object Triggers

Triggers are user-definable functions that validate or modify object or property state.

C.2.3.1. 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.

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.2.3.2. 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 into 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 into the data store. Typically used to transform an object just prior to its storage (for example, encryption).

C.2.3.3. 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 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 into 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 into the data store. Typically used to transform a given property prior to its object's storage.

C.2.3.4. Storage Trigger Sequences

The triggers are executed in the following orders.

Object Retrieval Sequence

1. Retrieve the raw object from the data store

2. Call object `onRetrieve` trigger
3. Per-property within the object (order undefined):
   - 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 (order undefined):
   - Call property `onValidate` trigger
   - Call object `onValidate` trigger
   - Perform schema validation (see the *Schema Validation* section)
2. Per-property trigger within the object (order undefined):
   - Call property `onStore` trigger
   - Call object `onStore` trigger
   - Store the object with any resulting changes to the data store

C.2.4. 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.2.5. 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.

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

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 (key word script), or the pointer to the file containing the script (key word file), to be executed.

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

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.2.6. Custom Managed Objects

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.2.6.1. 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.2.6.2. 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.2.6.3. 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.

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.2.6.3.1. Creating Objects

The following script example creates an object type.

```
openidm.create("managed/foobar/myidentifier", mymap)
```

C.2.6.3.2. 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.2.6.3.3. Patching Objects

You can partially update an object using a patch, which changes only the specified properties of the object. OpenIDM supports patch by query, so the caller need not 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":"/adPassword","value": "Password"}]'
 http://localhost:8080/openidm/managed/user?_action=patch&_query-id=for-username&uid=DDOE
```

For the syntax on how to formulate the query `_query-id=for-username&uid=DDOE` see Section C.2.6.3.6, "Querying Object Sets".

C.2.6.3.4. 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.2.6.3.5. Reading Objects

The following script example reads an object type.

```
val = openidm.read("managed/foobar/myidentifier")
```

C.2.6.3.6. Querying Object Sets

The following script example queries object type instances.

```
var params = {
    "_query-id": "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.2.7. 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`.

C.3. 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.3.1. 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.3.2. 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.3.3. 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.

C.3.4. 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.3.5. 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.3.6. 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.

C.3.7. Creating Objects

The following script example creates an object type.

```
openidm.create("config/custom/myconfig", mymap)
```

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.3.9. 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.3.10. Reading Objects

The following script example reads an object type.

```
val = openidm.read("config/custom/myconfig")
```

C.4. 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.5. Audit Objects

Audit objects house audit data selected for local storage in the OpenIDM repository. For details, see the chapter on *Using Audit Logs*.

C.6. Links

Link objects define relations between source objects and a 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.

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 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,
  "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
}
```

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"`.

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 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

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 chapter *Correlation* 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

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.

condition

script object, optional

A script to determine whether the mapping should be executed or not. The root scope contains the value of the source in the `"source"` property (if specified). 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
}
```

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.

D.1.2.1. Script Object

Script objects take the following form.

```
{
  "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.

```
{
   "_rev":string,
   "linkType":string,
   "secondId":string,
   "_id":string,
   "reconId":string,
   "firstId":string
}{
  "f": string,
  "targetId": string,
  "reconId" : string
}
```

_rev

string, required

The value of link object's revision.

linkType

string, required

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.

_id

string

The identifier of the link object.

reconId

string or null

The identifier of the last reconciliation job that processed this link. OpenIDM uses this during reconciliation to detect orphan source and target 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.

   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 target object already 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.

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
```

Response (success)

Example:

```
HTTP/1.1 204 No Content
...
```

Response (synchronization failure)

Example:

```
HTTP/1.1 409 Conflict
...
[JSON representation of error]
```

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
```

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.

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/1.1 204 No Content
ETag: "1"
...
```

Example Response: Updating an existing object (version conflict)

```
HTTP/1.1 409 Conflict
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 know 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 not provided, then the server selects its own identifier.

Example 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]
```

DELETE

Deletes a managed object.

Example Request

```
DELETE /openidm/managed/user/c3471805b60f
If-Match: "0"
...
```

Example Response (success)

```
HTTP/1.1 204 No Content
...
```

Example Response (version conflict)

```
HTTP/1.1 409 Conflict
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

```
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"
...
```

Example Response (version conflict)

```
HTTP/1.1 409 Conflict
Content-Type: application/json
Content-Length: 89
...

[JSON representation of error]
```

X-HTTP-Method-Override

If an HTTP client or server container does not support a particular method, a request can be submitted through POST with the `X-HTTP-Method-Override` header set to the intended method.

Example Request

```
POST /openidm/managed/user/5752c0fd9509 HTTP/1.1
X-HTTP-Method-Override: PATCH
Content-Type: application/patch+json
Content-Length: 456
If-Match: "0"
...

[JSON representation of patch document to apply]
```

Appendix F. Scripting Reference

Scripting lets you customize how OpenIDM works in various ways, such as providing custom logic between source and target mappings, defining correlation rules, filters, and triggers.

F.1. Configuration

You define scripts using script objects, which can either include the code directly in the configuration, or call an external file containing 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"`.

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 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 sets the `__PASSWORD__` attribute to `defaultpwd` when OpenIDM creates a target object.

```
"onCreate" : {
  "type"  : "text/javascript",
  "source": "target.__PASSWORD__ = 'defaultpwd'"
}
```

The following example 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

Functions (access to managed objects, system objects, configuration objects) within OpenIDM are accessible to scripts via the `openidm` object, which is included in the top-level scope provided to each script.

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

- A `null` value if successful.

Throws

- An exception is thrown if the object could not be created for any reason.

F.3.2. 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.3.3. openidm.update(id, rev, value)

This function updates a resource object.

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

- A `null` value if successful.

Throws

- An exception is thrown if the object could not be updated for any reason.

F.3.4. 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.

Returns

- A `null` value if successful.

Throws

- An exception is thrown if the object could not be deleted for any reason.

F.3.5. 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 parameters to pass to the query.

Returns

- The result of the query.

Throws

- An exception is thrown if the given query could not be processed for any reason.

F.3.6. openidm.action(id, params, value)

This function performs an action on the specified OpenIDM resource object.

Parameters

id

string

The identifier of the object to perform the action on.

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.3.7. 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.

Throws

- An exception is thrown if the object could not be encrypted for any reason.

F.3.8. 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.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`

onRetreive, onValidate

In `openidm/conf/router.json`

onRequest, onResponse

F.5. 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.

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.

Appendix G. Scheduler Reference

OpenIDM lets you schedule many tasks and events including reconciliation and synchronization, but also arbitrary events by use of scheduler objects.

OpenIDM supports **cron**-like syntax to schedule events and tasks, based on expressions supported by the Quartz Scheduler that OpenIDM bundles.

If you use configuration files to schedule tasks and events, then the scheduler files are located in the `openidm/conf` directory. By convention, OpenIDM uses file names of the form `scheduler-schedule-name.json`. OpenIDM dynamically picks up changes to scheduled tasks and events both, during initialization and also during runtime.

Scheduler Configuration Objects

Scheduler configuration objects take the following form.

```
{
 "enabled"       : true,
 "type"          : "cron",
 "startTime"     : "optional time",
 "endTime"       : "optional time",
 "schedule"      : "cron expression",
 "timeZone"      : "optional time zone",
 "invokeService" : "service identifier",
 "invokeContext" : "service specific context info"
}
```

The following simple example prints `Hello World` to the OpenIDM log, `/openidm/logs/openidm0.log.X`, each minute.

```
{
    "enabled": true,
    "type": "cron",
    "schedule": "0 0/1 * * * ?",
    "invokeService": "script",
    "invokeContext": {
        "script": {
            "type": "text/javascript",
            "source": "java.lang.System.out.println('Hello World');"
        },
        "input": {
            "edit": 26
        }
    }
}
```

Scheduler configuration objects take the following properties.

enabled

Set to `true` to enable the scheduler. When set to `false`, OpenIDM considers the scheduler configuration object dormant, and therefore does not let it be triggered or executed.

Rather than change the configuration or **cron** expressions, set `enabled` to `false` for task and event schedulers when you want to retain their configuration, but do not want them used.

type

Currently OpenIDM supports only `cron`.

startTime (optional)

Used to start the schedule some time in the future. If omitted or set to a time in the past, the task or event is scheduled starting immediately.

endTime (optional)

Used to plan the end of scheduling.

schedule

Takes **cron** expression syntax.

timeZone (optional)

If not set, OpenIDM uses the system time zone.

invokeService

Defines the type of scheduled event or action. OpenIDM accepts the following values.

- `provisioner`
- `script`

- `sync`

invokeContext

Specifies contextual information, such as the JavaScript to invoke. The discovery engine provides two mechanisms, reconciliation and synchronization. Both require that you specify the `invokeContext`.

The following example invokes reconciliation.

```
{
    "enabled": true,
    "type": "cron",
    "schedule": "0 0/1 * * * ?",
    "invokeService": "sync",
    "invokeContext": {
        "action": "reconcile",
        "mapping": "systemLdapAccount_managedUser"
    }
}
```

For reconciliation tasks, you can add the mapping definition inline rather than define it in `openidm/conf/sync.json`.

The following example shows a scheduler for LiveSync.

```
{
    "enabled": true,
    "type": "cron",
    "schedule": "0 0/1 * * * ?",
    "invokeService": "provisioner",
    "invokeContext": {
        "action": "liveSync",
        "source": "system/OpenDJ/__ACCOUNT__"
    }
}
```

Actual configuration differs from these examples depending on your configuration.

G.1. Scheduled Task Use Cases

OpenIDM lets you schedule not only reconciliation and synchronization, but also lets you use scheduling to trigger scripts, collect and run reports, trigger workflows, perform custom logging, and so forth. You can find a set of samples in the `openidm/samples/schedules` directory.

G.2. Cron Expressions

The Quartz Scheduler accepts expressions like those of the UNIX **cron** command. The syntax is documented extensively in the online Quartz *CronTrigger Tutorial*.

G.3. 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
```

G.4. 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.

Appendix H. 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.

H.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 they are specified in this array.

H.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
}
```

"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, then 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"`, then the other script(s) are executed. If not 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 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.

H.1.2. Script Execution Sequence

All "onRequest" and "onResponse" scripts are executed in a sorted sequence: first the "onRequest" scripts in a top down manner, then the "onResponse" in a bottom up way!

```
client -> filter 1 onRequest -> filter 2 onRequest -> resource
client <- filter 1 onResponse <- filter 2 onResponse <- resource
```

Here is an example of a `router.json` file and how the scritps 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');"
            }
        }
    ]
}
```

Will produce a log like:

```
requestFilter 1
requestFilter 2
responseFilter 2
responseFilter 1
```

H.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 context. Provided in 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.

```
{
  "error": integer,
  "reason": string,
  "detail": string
}
```

"error"

integer

The numeric code of the exception.

"reason"

string

The short reason phrase of the exception.

"detail"

string

The detailed message for the exception.

H.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"
            }
        }
    ]
}
```

Appendix I. Embedded Jetty Configuration

OpenIDM 2.0.3 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.

I.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.

I.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:

```
<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

I.1.2. Accessing Generic Properties

```
<Call class="org.forgerock.openidm.jetty.Param" name="getProperty">
  <Arg>org.forgerock.openidm.some.sample.property</Arg>
</Call>
```

I.2. Jetty Default Settings

By default the embedded Jetty server uses the following settings.

- An HTTP connector, listening on port 8080

- 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.

Index

T

W