PingIDM

June 5, 2025

PINGIDM
Version: 7.3

Table of Contents

Release notes

ForgeRock Identity Management (IDM) software provides centralized, simple management and synchronization of identities for users, devices, and things. IDM software is highly flexible and therefore able to fit almost any use case and workflow.

These release notes are written for anyone using the IDM 7.3 release. Read these notes before you install or upgrade ForgeRock Identity Management software.

What's New

New features and improvements in this version.

Prepare for Deployment

The requirements for running IDM software in production.

Compatibility

Key implementation changes and compatibility with previous deployments.

Bug Fixes

Bug fixes, limitations, and open issues.

Doc Updates

Documentation changes.

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

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

What's new

Maintenance releases

ForgeRock maintenance releases contain a collection of fixes and minor RFEs grouped together and released as part of our commitment to support our customers.

IDM 7.3.2 is the latest release targeted for IDM 7.3 deployments and can be downloaded from the Backstage Download Center.

> **Note**
>
> You can deploy the release as an initial deployment or as an update from an existing 7.3.x deployment. Learn more about updating from 7.3.x in Update to a maintenance release.

IDM 7.3.2 features

Secure RCS access

You can create stricter RCS authorization and access rules. To enable authorization for RCS, add an appropriate role to the static-user mapping used for the RCS subject and write the appropriate access rules to permit this role to be granted access to the `openicf` servlet on the path (pattern) corresponding to the RCS name used in the RCS configuration.

Learn more in Secure RCS access.

Array comparison

You can choose how synchronization detects managed object array changes using *unordered* or *ordered* comparison using the configuration property `comparison` in the schema. Unordered JSON array comparison ignores the order of elements and can negate the need for certain custom scripts within mappings.

Learn more about managed object schema properties and array comparison.

`_api` parameter requires authorization

Requests passing the `_api` parameter now require authorization. Learn more in Common REST.

Jetty 12 support

The embedded Jetty web server supports Jetty 12.

Java 17 support

This IDM release requires Java 17. Learn more in Embedded Jetty configuration.

IDM 7.3.1 features

Workflow engine upgrade

The Flowable embedded workflow engine has been upgraded to version 6.8.0. If you're upgrading from a previous version of IDM and use workflow, this upgrade requires one or more incremental upgrade scripts. For more information, refer to Upgrade an existing repository.

End User UI supports array properties

Array properties now display in the End User UI.

IDM 7.3.0 features

Support for Bouncy Castle FIPS

IDM now supports the use of Bouncy Castle FIPS as a security provider. Bouncy Castle FIPS is useful when dealing with government data, where meeting the FIPS 140-2 security requirement is necessary for regulatory compliance.

For information on how to configure Bouncy Castle, refer to FIPS 140-2 compliance.

Support for UTF-8 email addresses

IDM now supports UTF-8 (non-ASCII/international) characters in email addresses, such as **zoë@example.com**. When sending emails to these type of addresses, the configured SMTP server must also support UTF-8.

Disable delegated administrator sort and filter while searching

You can now disable delegated administrator sort and filter while searching resource collections in the End User UI. For more information, refer to Disable sort and filter for resource collections.

Workflows now support JavaScript

IDM workflows now support JavaScript in addition to Groovy. For more information about scripting workflows, refer to BPMN 2.0 and workflow tools.

Patch operation improvements

It is now possible to patch the root of an object. The only supported patch operations on the root of an object are `remove` and `replace`.

Improvements to the /system endpoint

`/system` endpoints now support specifying additional fields when also using `*`. This allows callers to get fields that are not returned by default.

New sync mapping configuration fields

New sync mapping configuration fields, `defaultSourceFields` and `defaultTargetFields`, allow specifying which fields to use for read and query requests made on source and target resource collections.

Security advisories

ForgeRock issues security advisories in collaboration with our customers and the open source community to address any security vulnerabilities transparently and rapidly. ForgeRock's security advisory policy governs the process on how security issues are submitted, received, and evaluated as well as the timeline for the issuance of security advisories and patches.

For details of all the security advisories across ForgeRock products, refer to Security Advisories in the *Knowledge Base library*.

Before you install

This section covers requirements before you run ForgeRock Identity Management software, especially in a production environment. If you have a special request to support a component or combination not listed here, contact ForgeRock at info@forgerock.com.

Hardware and memory requirements

Due to the underlying Java platform, IDM software runs well on a variety of processor architectures.

When you install IDM for evaluation with the embedded DS repository, you need:

- 256 MB memory (32-bit) or 1 GB memory (64-bit) available.
- 10 GB free disk space for the software and sample data.

> **Important**
>
> A DS repository (whether embedded or external) requires free disk space of 5% of the filesystem size, plus 1 GB by default. To change this requirement, set the `disk-full-threshold` in the DS configuration. For more information, refer to Disk Space Thresholds in the *DS Maintenance Guide*.
> In the case of an embedded DS instance, you can manage the configuration using the `dsconfig` command in `/path/to/openidm/db/openidm/opendj/bin` .

In production, disk space and memory requirements depend on the size of your external repository, as well as the size of the audit and service log files that IDM creates.

The amount of memory that IDM consumes is highly dependent on the data that it holds. Queries that return large data sets will have a significant impact on heap requirements, particularly if they are run in parallel with other large data requests. To avoid out-of-memory errors, analyze your data requirements, set the heap configuration appropriately, and modify access controls to restrict requests on large data sets.

IDM exposes many JVM metrics to help you analyze the amount of memory that it is consuming. For more information on analyzing hardware and memory performance, see Load testing.

Operating System requirements

IDM 7.3 software is supported on the following operating systems:

- Red Hat Enterprise Linux (and Rocky Linux) 7.9, 8.7, and 9.1
- Ubuntu Linux 20.04 and 22.04

- Windows Server 2019 and 2022

Java requirements

IDM software supports the following Java environments:

Supported Java Versions

Vendor	Versions
OpenJDK, including OpenJDK-based distributions: • AdoptOpenJDK/Eclipse Temurin • Amazon Corretto • Azul Zulu • Red Hat OpenJDK **Note** ForgeRock tests most extensively with AdoptOpenJDK/Eclipse Temurin. ForgeRock recommends using the HotSpot JVM.	17*
Oracle Java	17*

* Version 17.0.9 or higher.

Tip

ForgeRock recommends that you keep your Java installation up to date with the latest security fixes.

Supported web application containers

You must install IDM as a standalone service, using the bundled Apache Felix framework and Jetty web application container. Alternate containers are not supported. IDM bundles Jetty version 12.0.19.

Supported repositories

The following repositories are supported for use in production:

- ForgeRock Directory Services (DS) 7.2, 7.3, and 7.4.

  By default, IDM uses an *embedded* DS instance for testing purposes. The embedded instance is not supported in production. If you want to use DS as a repository in production, you must set up an external instance.

- MySQL version 5.7 and 8.0 with MySQL JDBC Driver Connector/J 8.0.

  Important

  Do not use Connector/J versions 8.0.23 through 8.0.25. Why?

- MariaDB version 10.6.11 and 10.10.2 with MySQL JDBC Driver Connector/J 8.0.

Important

Do not use Connector/J versions 8.0.23 through 8.0.25. Why?

- Microsoft SQL Server 2019 and 2022.
- Oracle Database 19c and 21c.
- PostgreSQL 13.10, 14.7, and 15.2.
- IBM DB2 11.5.

ForgeRock supports repositories in cloud-hosted environments, such as AWS and GKE Cloud, as long as the underlying repository is supported. In other words, the repositories listed above are supported, regardless of how they are hosted.

Note

These repositories might not be supported on all operating system platforms. refer to the specific repository documentation for more information.
Do not mix and match versions. For example, if you are running Oracle Database 19c, and want to take advantage of the support for Oracle UCP, download driver and companion JARs for Oracle version 19c.

Supported browsers

The IDM UI has been tested with the latest, stable versions of the following browsers:

- Chrome and Chromium
- Edge
- Firefox
- Safari

Supported connectors

IDM bundles the following connectors:

- Adobe Cloud Marketing connector
- CSV File connector
- Database Table connector
- Google Apps connector
- Groovy Connector Toolkit

  This toolkit lets you create scripted connectors to virtually any resource.

- Kerberos connector

  The Kerberos connector bundled with IDM 8 is *not* backward-compatible with IDM 6.x. IDM 8 uses Groovy version 3.0. IDM 6.5 uses version 2.5, and IDM 6 uses version 2.4. The bundled Kerberos connector requires Groovy version 3.0.

- LDAP connector

  Using the LDAP connector to provision to Active Directory is supported with Active Directory Domain Controllers, Active Directory Global Catalogues, and Active Directory Lightweight Directory Services (LDS).

- Marketo connector
- MongoDB connector
- Microsoft Graph API connector
- Salesforce connector
- SCIM connector
- Scripted REST connector

  The scripted REST connector bundled with IDM 8 is *not* backward-compatible with IDM 6.x. IDM 8 uses Groovy version 3.0. IDM 6.5 uses version 2.5, and IDM 6 uses version 2.4. The bundled scripted REST connector requires Groovy version 3.0.

- Scripted SQL connector

  The scripted SQL connector bundled with IDM 8 is *not* backward-compatible with IDM 6.x. IDM 8 uses Groovy version 3.0. IDM 6.5 uses version 2.5, and IDM 6 uses version 2.4. The bundled scripted SQL connector requires Groovy version 3.0.

- ServiceNow connector
- Scripted SSH connector

  The scripted SSH connector bundled with IDM 8 is *not* backward-compatible with IDM 6.x. IDM 8 uses Groovy version 3.0. IDM 6.5 uses version 2.5, and IDM 6 uses version 2.4. The bundled scripted SSH connector requires Groovy version 3.0.

Additional connectors are available from the Backstage download site.

A PowerShell Connector Toolkit is bundled with the .NET remove connector server. This toolkit lets you create scripted connectors to address the requirements of your Microsoft Windows ecosystem.

Windows Server 2012 R2, 2016, and 2019 are supported as the remote systems for connectors and password synchronization plugins.

You must use the supported versions of the .NET Remote Connector Server (RCS), or the Java Remote Connector Server (RCS). The 1.5.x Java RCS is backward-compatible with the version 1.1.x connectors. The 1.5.x .NET RCS is compatible only with the 1.4.x and 1.5.x connectors. For more information, refer to IDM / OpenICF Compatibility Matrix.

Important

In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
Learn more in What's new.

The Java RCS requires Java 17 and is supported on any platform on which Java runs.

The .NET RCS requires the .NET framework (version 4.6.2 or later) and is supported on Windows Server versions 2012 R2, 2016, and 2019.

> **Important**
>
> Although the scripted connector toolkits are supported, connectors that you build with these toolkits are not supported. You can find examples of how to build connectors with these toolkits in Samples.

The following table lists the connector and RCS versions that are supported across IDM versions. For a list of connectors supported with this IDM release, refer to the OpenICF connector documentation. For a list of connector releases associated with this version of IDM, refer to the OpenICF release notes.

IDM / OpenICF Compatibility Matrix

IDM Version	RCS Version	Java Connectors	Scripted Groovy Connectors	.NET Connectors
4.x	1.4.x, 1.5.x	Java connectors version 1.1.x - 1.5.x	Scripted REST, Scripted CREST, Scripted SQL, SSH, Kerberos connectors *up to* version 1.5.1.0.	PowerShell Connector 1.4.x
5.x	1.4.x, 1.5.x	Java connectors version 1.1.x - 1.5.x	Scripted REST, Scripted CREST, Scripted SQL, SSH, Kerberos connectors *up to* version 1.5.1.0.	PowerShell Connector 1.4.x
6.x	1.4.x, 1.5.x	Java connectors version 1.1.x - 1.5.x	Scripted REST, Scripted CREST, Scripted SQL, SSH, Kerberos connectors *up to* version 1.5.1.0.	PowerShell Connector 1.4.x
7.x	1.4.x, 1.5.x	Java connectors version 1.1.x - 1.5.x	Scripted REST, Scripted SQL, SSH, Kerberos connectors version 1.5.x.	PowerShell Connector 1.4.x, 1.5.x

Supported password synchronization plugins

The following table lists the supported password synchronization plugins:

Plugin	Supported Version
DS Password Synchronization Plugin	7.3.x, supported with DS 7.3.x and IDM 7.3.x 7.1.x, supported with DS 7.1.x, DS 7.2.x, IDM 7.1.x, and IDM 7.2.x 7.0.1, supported with DS 7.0.x, IDM 7.0.x, and IDM 7.1.x 6.5.0, supported with DS 6.5.x and IDM 6.5.x 6.0, supported with DS 6.0.x and IDM 6.0.x 5.5.0, supported with DS 5.5.x and IDM 5.5.x 5.0, supported with DS 5.0.x and IDM 5.0.x 3.5, supported with OpenDJ 3.5 and OpenIDM 4.x DS Password Sync plugins are not supported with DS OEM
Active Directory Password Synchronization Plugin	1.7.0 and 1.5.0 supported on Windows Server versions 2012 R2, 2016, 2019, and 2022

Third-Party software

ForgeRock provides support for using the following third-party software when logging ForgeRock Common Audit events:

Software	Version
Java Message Service (JMS)	2.0 API
MySQL JDBC Driver Connector/J	8 (at least 8.0.19) **Important** Do not use Connector/J versions 8.0.23 through 8.0.25. Why?
Splunk	8.0 (at least 8.0.2)

Tip

Elasticsearch and Splunk have native or third-party tools to collect, transform, and route logs. Examples include Logstash and Fluentd.
ForgeRock recommends that you consider these alternatives. These tools have advanced, specialized features focused on getting log data into the target system. They decouple the solution from the ForgeRock Identity Platform systems and version, and provide inherent persistence and reliability. You can configure the tools to avoid losing audit messages if a ForgeRock Identity Platform service goes offline, or delivery issues occur.
These tools can work with ForgeRock Common Audit logging:

- Configure the server to log messages to standard output, and route from there.
- Configure the server to log to files, and use log collection and routing for the log files.

Although ForgeRock does not provide support for these tools, you can any use of the following third-party software to monitor ForgeRock servers:

Software	Version
Grafana	5 (at least 5.0.2)
Graphite	1
Prometheus	2.0

For Hardware Security Module (HSM) support, ForgeRock software requires a client library that conforms to the PKCS#11 standard v2.20 or later.

Incompatible changes

When you update to IDM 8.0.0 from the last major version, the following changes may impact existing deployments. Adjust existing scripts, files, clients, and so on, as necessary.

If you're upgrading from an older release, review the changed functionality from all releases after your current version of IDM:

- Incompatible changes (7.2.x)
- Incompatible changes (7.1.x)
- Incompatible changes (7.0.x)

Changes between IDM 7.3.1 and 7.3.2

`_api` parameter requires authorization

Requests passing the `_api` parameter now require authorization. Learn more in Common REST.

Array comparison

Starting with IDM 7.3.0, unordered array comparison became the default behavior. For this release of IDM, ordered array comparison is the default behavior, restoring the default behavior from prior to IDM 7.3.0.

You can now use the `comparison` managed object schema configuration property to choose how JSON array comparisons are made with regard to array order.

Learn more about managed object schema properties and array comparison.

Java upgrade

You must upgrade to Java 17, which is required by Jetty 12, to run IDM 7.3.2. Learn more in Embedded Jetty configuration.

Changes between IDM 7.3.0 and 7.3.1

Workflow engine upgrade

The Flowable embedded workflow engine has been upgraded to version 6.8.0. If you're upgrading from a previous version of IDM and use workflow, this upgrade requires one or more incremental upgrade scripts. For more information, refer to Upgrade an existing repository.

Changes between IDM 7.2.x and 7.3.0

Synchronization JSON array comparison is order-agnostic

JSON array comparison during sync is now *order-agnostic*. This change may negate the need for certain custom scripts within mappings. For example, scripts that were previously required to sort `ldapGroups` values to avoid unnecessary target object updates.

Attribute encryption on assignments

Assignment attributes are now encrypted if the corresponding connector attribute indicates confidentiality, based on the attribute's `nativeType` (such as `JAVA_TYPE_GUARDEDSTRING` or `JAVA_TYPE_GUARDED_BYTE_ARRAY`). As part of this change, the managed assignment object now includes the following property:

```
"attributeEncryption" : { }
```

If `attributeEncryption` is not present, the assignment attributes are not encrypted. If the property is present but empty, it will default to IDM's default encryption cipher. To specify a different cipher, add the `cipher` property. For example:

```
"attributeEncryption" : {
  "cipher" : "AES/CBC/PKCS5Padding"
}
```

Additionally, `secrets.json` has a new secret: `idm.assignment.attribute.encryption`.

Deprecation

The following features are deprecated and likely to be discontinued in a future release.

Social authentication

Social authentication is deprecated and will be removed in a future release of IDM. The feature will be a function of AM. Once a user has logged in through AM (using a social provider or some other way), they can obtain an access token with that session and use the access token to interact with IDM through the rsFilter configuration.

Additionally, Microsoft has deprecated the "Sign In with LinkedIn" functionality as of August 1, 2023. Refer to Sign In with LinkedIn.

Access configuration in access.js

In previous releases, access rules were configured in the `access.js` script. This script has been replaced by an `access.json` configuration file, that performs the same function. Existing deployments that use customized `access.js` files are still supported for backward compatibility. However, support for access rules defined in `access.js` is deprecated, and will be removed in a future release. You should move these access rules to a `conf/access.json` file. For more information, refer to Authorization and roles.

Actions on scheduler endpoint

The `action` parameter on the `scheduler` endpoint was deprecated in Version 1 of the endpoint and is not supported in Version 2.

To validate a cron expression, use the `validateQuartzCronExpression` action on the `scheduler/job` endpoint, as described in Validate Cron Trigger Expressions.

Health endpoints

The `health` endpoints, used to monitor system activity have been deprecated in this release, as their functionality was not considered to be of much use.

The information available on `health/recon` was node-specific. Instead, you can retrieve cluster-wide reconciliation details with a GET on the `recon` endpoint.

The information available on the `health/os` and `health/memory` endpoints can be retrieved by inspecting the JVM metrics.

Conditional query filters

The syntax of conditional query filters and scripts within notification filters has changed in this release. In previous IDM releases, request properties such as `content` in create and update requests or `patchOperations` in patch requests were referenced directly. For example, the `notification-newReport.json` configuration previously used the following query filter:

```
"condition" : "content/manager pr"
```

In IDM 7, query filters and scripts should reference the `request` object to obtain any request properties. Sample query filters have been changed accordingly. For example, the query filter in `notification-newReport.json` has been changed to the following:

```
"condition" : "request/content/manager pr",
```

This syntax is more verbose, but it lets script implementations use request visitors logic based on the request type, and is more consistent with generic router filters.

The old request syntax will still work in IDM 7.0, but is considered deprecated. Support for the old syntax will be removed in a future release. Note that this change is limited to notification filters. Filters such as those used with scripted endpoints have never supported direct access to request properties, and are therefore not changing. For more information on notification filters, refer to Configure notifications.

Self-Service stages

Self-Service Stages (described in Self-service stage reference) are deprecated in this release and support for their use will be removed in a future release. From IDM 7 onwards, this functionality is replaced by AM Authentication Trees.

oauthReturn endpoint

Support for `oauthReturn` as an endpoint for OAuth2 and OpenID Connect standards has been deprecated for interactions with AM and will be removed in a future release. Support for interactions with social identity providers was removed in IDM 6.5.0.

Default versions of relevant configuration files no longer include `oauthReturn` in the `redirectUri` setting. However, for IDM 7.3, these configuration files should still work both with and without `oauthReturn` in the endpoint.

`timeZone` in schedules

In Configure schedules, setting a time zone using the `timeZone` field is deprecated. To specify a time zone for schedules, use the `startTime` and `endTime` fields.

MD5 and SHA-1 hash algorithms

Support for the `MD5` and `SHA-1` hash algorithms is deprecated and will be removed in a future release. You should use more secure algorithms in a production environment. For a list of supported hash algorithms, refer to Salted Hash Algorithms.

`JAVA_TYPE_DATE` attribute type

Support for the native attribute type, `JAVA_TYPE_DATE`, is deprecated and will be removed in a future release. This property-level extension is an alias for `string`. Any dates assigned to this extension should be formatted per ISO 8601.

POST request with `?_action=patch`

Support for a POST request with `?_action=patch` is deprecated, when patching a specific resource. You can still use `?_action=patch` when patching by query on a collection.

Clients that do not support the regular PATCH verb should use the `X-HTTP-Method-Override` header instead.

For example, the following POST request uses the `X-HTTP-Method-Override` header to patch user jdoe's entry:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--header "X-HTTP-Method-Override: PATCH" \
--data '[
    {
        "operation":"replace",
        "field":"/description",
        "value":"The new description for Jdoe"
    }
]' \
"http://localhost:8080/openidm/managed/user/jdoe"
```

`minLength` property

The managed object property `minLength` is deprecated. When you need to specify a minimum length for a property, use the `minimum-length` policy:

```
{
    "policyId" : "minimum-length",
    "params" : {
        "minLength" : 8
    }
}
```

Splunk and Elasticsearch audit handlers

The Splunk and Elasticsearch audit event handlers are deprecated and will be removed in a later release.

IDM 7.3 supports both file-based audit handlers and logging to standard output, which Elasticsearch and Splunk can consume.

Read requests at top of `/config`

Support for top-level read requests to the `/config` endpoint is deprecated. You can still retrieve a list of config IDs by querying the `/config` endpoint.

Defining object schema `type` attribute in an array when it is a single type

Support for specifying an object's schema `type` attribute in an array when there is only a single type is deprecated and will be removed in a later release.

This affects schemas with `type` attribute definitions in the form:

```
{
    "type" : ["string"]
}
```

`type` attribute definitions in this form should be updated to:

```
{
    "type" : "string"
}
```

For additional information, refer to the JSON schema `type` attribute definition.

Discontinued

The following features or functionalities were removed in this release.

IDM 7.3.2 removals

Apache Felix web console

We've removed the Apache Felix web console in this release of IDM.

Java 11 support

Java 11 support has been removed from this release. You must upgrade to Java 17, which is required by Jetty 12 to run IDM 7.3.2. Learn more in Embedded Jetty configuration.

Gzip handler `compressionLevel` and `excludedAgentPatterns` properties

In Jetty 12, the `compressionLevel` and `excludedAgentPatterns` properties have been removed from the Gzip handler.

IDM 7.3.1 removals

No features or functionality were removed in this release.

IDM 7.3.0 removals

No features or functionality were removed in this release.

Fixed issues

IDM 7.3.2

The following important bugs were fixed in this release:

- OPENIDM-18495: Admin UI: Connector Data Tab is sending a queryFilter with bad sortKeys
- OPENIDM-18848: New string and number attributes added to managed object schema default to "searchable"
- OPENIDM-19405: Special characters (non-ASCII) inside of emails being sent from IDM/Identity Cloud fail
- OPENIDM-19666: Admin UI should not inject domain configuration property within Connector config for GoogleApps/ Salesforce
- OPENIDM-19755: GoogleApps Connector: update sample to reflect replacement of __SECONDARY_EMAIL__ by __SECONDARY_EMAILS__
- OPENIDM-19829: Time spent in authentication service is not included in access audit elapsed time
- OPENIDM-19879: Query additional recon source/target pages whenever a paging cookie is returned irrespective of whether paging is enabled
- OPENIDM-19918: Order-agnostic comparison of array fields in sync must be optional
- OPENIDM-20063: Syncing ldap object with `//` in them does not work
- OPENIDM-20142: Permanent failure caused by transient connector validation failure during provisioner service activation
- OPENIDM-20238: SynchronizationException, Expecting a Map or List
- OPENIDM-20337: Provisioner createCoreConfig action should omit poolConfigOption properties for non-pooleable Connectors

IDM 7.3.1

The following important bugs were fixed in this release:

- OPENIDM-19467: Sync initialization mapping failures in one mapping will not disrupt other mappings from initializing
- OPENIDM-19328: Queued Sync does not recover intermittently following node restart
- OPENIDM-19192: Edit Personal Info required to be read-only not working as expected
- OPENIDM-19141: Honor the tablePrefix and tablePrefixIsSchema configuration options that allow the customer to prefix tables for workflow
- OPENIDM-18875: Incorrect behavior in handling variables in workflow subprocesses
- OPENIDM-18613: Setting external system's userPassword to null does not remove userPassword attribute when nativeName is `__PASSWORD__`
- OPENIDM-17481: Managed object schema can now describe a field as a nullable array and specify a default value for this field if not provided in a create request

IDM 7.3.0

The following important bugs were fixed in this release:

- OPENIDM-18895: ManagedObjectSet patch contract lacks proper MVCC retry
- OPENIDM-18875: Incorrect behavior in handling variables in workflow subprocesses
- OPENIDM-18870: No ability to delete an inline reconciliation or schedule script
- OPENIDM-18868: Inability to save a schedule when you add or remove a passed variable
- OPENIDM-18865: Script changes cannot be saved unless you click outside of the Inline Script box
- OPENIDM-18831: Order agnostic JsonValue comparisons necessary in sync
- OPENIDM-18827: Delegated Admin UI - option to disable relationshipArray grid sorting and searching
- OPENIDM-18823: Explicitly ask for all non-reference fields when presented with '*'
- OPENIDM-18807: IDM sample "Provision user with workflow" is not working as expected
- OPENIDM-18806: SpecReference caused regressions for explicit
- OPENIDM-18794: queryFilter should not be transformed if already transformed
- OPENIDM-18779: Legal noticed disappeared from OpenIDM zip
- OPENIDM-18776: Sync operation fails silently without error when linkQualifier script returns wrong data type
- OPENIDM-18753: Fields on a non-configured relationship request no longer returning
- OPENIDM-18739: Authz - own relationship query is outdated
- OPENIDM-18707: JsonUtil.jsonValueIsEqualWithoutRespectingOrder does not work when consumed via javascript
- OPENIDM-18656: Read for singleton reverse spec reference is not working
- OPENIDM-18629: Clustered recon source page jobs should use nanotime within job identifier
- OPENIDM-18625: Top-level router contains route to empty subrouter on route deregistration
- OPENIDM-18580: It’s not possible to type in a Base DN containing a space in IDM native console.
- OPENIDM-18544: AD User with a Manager Cannot Update Manger in IDM
- OPENIDM-18509: AM is unable to list groups created by IDM if DS repo is restarted.
- OPENIDM-18506: IDC’s internal.json should be in conf directory
- OPENIDM-18498: Queued Sync not triggered if target is a CREST proxy endpoint
- OPENIDM-18483: Add "name" to resourceCollection query fields for Platform and FeatureService Groups
- OPENIDM-18476: Changing a managed object field of type `number` results in a default value of 0 if not set
- OPENIDM-18444: MVCC semantics not enforced during target update synchronization operations
- OPENIDM-18414: Error in pwpolicy.js from multiple-passwords sample

- OPENIDM-18411: RDVP values can be removed upon signal receipt when multiple multi-traversal RDVPs have matching initial traversal relationship
- OPENIDM-18388: ClusteredReconWatchdog will incorrectly schedule sourcePageCompletionCheck jobs for a reconById recon running against a mapping configured for clustered recon
- OPENIDM-18360: One-to-many relationship not enforced when delegated admin has no openidm-admin role
- OPENIDM-18336: "managed/assignment" missing the "condition" property in default DS repo config
- OPENIDM-18335: Assignment processing can mutate source effective assignments resulting in incorrect lastSync state
- OPENIDM-18272: Save managed object properties correctly in Identity Management native console
- OPENIDM-18247: get_target_preview_external_user_provisioned_linked_mapping test is failing with "Expected a single link, found 0"
- OPENIDM-18243: Connector names need to be validated as alpha-numeric in UI
- OPENIDM-18238: clustered recon: schedule creation in response to orphaned job may incorrectly propagate source pages, resulting in 'hung' recon
- OPENIDM-18192: Virtual property is removed when another Virtual property is updated
- OPENIDM-18167: mergeWithTarget assignment operation handles previously replaced object incorrectly
- OPENIDM-18153: Throw statement truncates user-defined exception
- OPENIDM-18149: Relationship entry needs to be selected two times to see the "Remove" option in End User UI
- OPENIDM-18138: Setting empty conditional grants on 'old' object state causes all conditional relationships to be queried during RDVP calculations for both managed object update and signal receipt
- OPENIDM-18123: Correctly load scripts that use ISO 8859-1 encoding
- OPENIDM-18077: The CANNOT_CONTAIN_OTHERS password policy in IDM is case sensitive.
- OPENIDM-18067: SourcePageToken equals, toString, and hashCode incomplete
- OPENIDM-18064: ReconCancellation initialization should handle the activation→deactivation→activation of the ReconciliationService
- OPENIDM-18001: Locale codes not working correctly in email templates
- OPENIDM-17980: Inconsistent Policy Validation message on Admin UI for some policyId's
- OPENIDM-17954: POST _action=create for undefined resource collection results in internal server error
- OPENIDM-17937: Recon query retry value should be increased to a total span > (rcs_staggered_connection_creator_interval + rcs_houskeeping_interval)
- OPENIDM-17900: Workday connector fails to start
- OPENIDM-17894: 404 page license is three years out of date
- OPENIDM-17837: Unable to index nested arrays with JDBC repos
- OPENIDM-17825: JsonValuePatch throws an NPE when patching a subject missing a field used in the complex filter

- OPENIDM-17771: Processing of misfired triggers eventually leads to failure of all scheduled tasks
- OPENIDM-17750: From field not allowing saving email address with multiple "domains" after the @
- OPENIDM-17707: The Connector UI "Object Classes to Synchronize" parameter is storing values incorrectly
- OPENIDM-17664: Adding whitespace in BaseDN results in invalid configuration
- OPENIDM-17642: Document the usage of cancel action on openidm.action "recon"
- OPENIDM-17612: Incorrect relationship collection query results with _sortKeys=_id
- OPENIDM-17556: Executing REST PUT against a managed object without conditional roles will erase all object RDVPs
- OPENIDM-17533: Allow configuration changes to the repo.ds.json file to take effect without restarting IDM
- OPENIDM-17531: Conditional policy is not enforced for patch remove
- OPENIDM-17529: LiveSync schedules are not saving correctly on first save
- OPENIDM-17483: Quotation marks is automatically removed from Query field of Role's condition
- OPENIDM-17200: ReconAssociation query with queryMissingSide=true and _fields params results in 500 error
- OPENIDM-17024: Admin UI - Query condition memberOfOrgIDs value not saved as a string
- OPENIDM-16830: fr-idm-managed-organization-name is not indexed
- OPENIDM-16768: Workflow process form should submit formProperty id instead of name
- OPENIDM-16725: managed.json updated incorrectly when relationship property is modified in the UI
- OPENIDM-16641: UI: Legacy Admin - config logic field "deleteQueryConfig" is leaking into UI generated managed config
- OPENIDM-15303: Scheduler is logging incorrect messages in openidm.log
- OPENIDM-15132: OPENIDM-14434 caused significant performance degradations
- OPENIDM-14666: SCIM connector cannot be configured through the UI
- OPENIDM-13209: Sorting is not working for edge_vertex query with embedded_dj repo

Limitations

ForgeRock Identity Management 7.3 has the following known limitations:

Workflow limitations

- Workflows are not supported with a DS repository. If you are using a DS repository for IDM data, you must configure a separate JDBC repository as the workflow datasource.
- The embedded workflow and business process engine is based on Flowable and the Business Process and Notation (BPMN) 2.0 standard. As an embedded system, local integration is supported. Remote integration is not currently supported.

Queries with a DS repository

For DS repositories, relationships must be defined in the repository configuration (`repo.ds.json`). If you do not explicitly define relationships in the repository configuration, you will be able to query those relationships, but filtering and sorting on those queries will not work. For more information, refer to Relationship Properties in a DS Repository.

Queries with an OracleDB repository

For OracleDB repositories, queries that use the `queryFilter` syntax do not work on CLOB columns in explicit tables.

Queries with privileges

Query filters used for privileges can only reference *direct* attributes of the object. For example, relationship fields cannot be referenced in a privilege filter.

Connector limitations

- When you add or edit a connector through the admin UI, the list of required `Base Connector Details` is not necessarily accurate for your deployment. Some of these details might be required for specific deployment scenarios only. If you need a connector configuration where not all the Base Connector Details are required, you must create your connector configuration file over REST or by editing the provisioner file. For more information, refer to Configure connectors.

`If-Match` requests

A conditional GET request, with the `If-Match` request header, is not supported.

Known issues

This topic lists important issues that remain open at the time of release.

IDM issues

- OPENIDM-848: Conflicting behavior might be observed between the default fields set by the onCreate script and policy enforcement
- OPENIDM-10490: admin UI doesn't allow multiple values for the objectClassesToSynchronize LDAP connector property
- OPENIDM-12540: Unable to change openidm-admin password via self service UI
- OPENIDM-13198: PATCH requests are transformed to UPDATE requests internally, affecting more attributes than they should
- OPENIDM-13592: optimize java script context caching to reduce transient memory allocation
- OPENIDM-14828: updateLastSync sets returnByDefault relationship to empty array
- OPENIDM-15376: Sorting on retries on Workflow deadletter jobs causes 500 error
- OPENIDM-15614: large group membership UPDATE/GET operations is slow against AD

- OPENIDM-15729: LastSync functionality is tightly coupled to the managed/user resource path
- OPENIDM-15810: CSV Bulk Upload intermittently fails to import users with Oracle explicit table
- OPENIDM-16224: Delegated admin doesn’t work for user who registers and logs in with Google idP
- OPENIDM-16228: Temporal Roles not showing in Admin UI w/DS as ID Repo
- OPENIDM-16250: Rhino scripts: resourceName.leaf() should be a string
- OPENIDM-16269: Rhino: lodash isEqual() always returns false for objects
- OPENIDM-16349: adpowershell provisioner account schema causes query with sortKey=distinguishedName to fail
- OPENIDM-16491: connection between agent and IDM/RCS would break after IDM pod relocated from one node to another
- OPENIDM-16516: Incoherent script hooks bindings when PATCH a relationship collection containing relationship properties
- OPENIDM-16697: Using Postgres and CITEXT, a user is unable to log in due to case sensitivity
- OPENIDM-16843: Relationships, having "returnByDefault=true" flag set, are not being included in oldObject/newObject values at onUpdate() trigger level when "_fields" is specified
- OPENIDM-17190: PBKDF2 pre-hashed passwords from IDM not working on DS
- OPENIDM-17327: Property Value Substitution failing for LoginURL in Salesforce Connector
- OPENIDM-17347: 500 RuntimeException when parsing some date formats in audit query
- OPENIDM-17443: Clean-up and remove obsolete nodes that appear "running" on the Cluster Node Status WIDGET?
- OPENIDM-17448: Incorrect Year Display with different timezone on Audit Events Dashboard
- OPENIDM-17466: Unit tests in ManagedObjectSetTest make false assumptions
- OPENIDM-17476: Missing matchAttribute property when using /openidm/config/fieldPolicy/ to configure password validator results in unexpected behaviour
- OPENIDM-17478: RDVP calculation does not respect the 'validate' config that can be disabled in managed.json
- OPENIDM-17488: Removing a parent relationship from a child org as owner/admin of that parent org returns a 404 instead of a 200 on JDBC/MySQL as repo
- OPENIDM-17516: Pattern policy ignored when doing operation replace with empty values
- OPENIDM-17630: A value set to the List of Names to Filter setting of a Provisioner via the UI disappears when saved and the provisioner is accessed again
- OPENIDM-17631: Overriding the key “aliases” in conf/secrets.json using $array and $list coercion type to support multiple key aliases is not working
- OPENIDM-17671: Request for postSync script hook
- OPENIDM-17760: "In" clause can not be called from javascript with openidm.query()
- OPENIDM-17813: File content incorrect on read

- OPENIDM-17815: Saving invalid script in managed.json causes managed object to return 404
- OPENIDM-17922: Sample scripted powershell with ad is missing ResolveUsername script
- OPENIDM-17983: Workflow process definition diagram is not displayed in the Admin UI
- OPENIDM-17997: Array virtual properties fail to update during a compound replace operation when revision data is included.
- OPENIDM-18039: Modify GroovyScript to utilize similar logic that RhinoScript is using in ScriptableWithDeferredBinding
- OPENIDM-18074: End-User UI Preferences property to READ-ONLY (Non-editable) not working
- OPENIDM-18132: Upgrade Rhino to resolve Issue #1232
- OPENIDM-18154: Mapping will restore itself after being deleted when moving position in grid holder view
- OPENIDM-18162: Transformation script for relationship attributes does not run in IDM 7.2.0
- OPENIDM-18196: Assignments with multivalued attributes triggers unnecessary updates on target objects
- OPENIDM-18218: RDVP and conditional grantee 'merry-go-round' causing superfluous relationship field reads
- OPENIDM-18231: Disabling and enabling livesync schedule changes value of source
- OPENIDM-18271: Adding Policy via UI doesn’t always work
- OPENIDM-18277: Task Scanner fails on erroneous conditional policy validation failure
- OPENIDM-18290: Dependent conditional policy not run when patching a property
- OPENIDM-18333: Policy validation does not work fine if values are provided to all fields together which are being used in policy validation
- OPENIDM-18340: Multi-language support for platform deployment is missing
- OPENIDM-18412: Value for boolean property in Linked Systems tab appears to be hidden
- OPENIDM-18493: Response from csv/template endpoint is different in IDM CDK
- OPENIDM-18496: Missing UI templates for Groovy scripted connectors 1.5
- OPENIDM-18643: Sporadic NPE upon Activation of the OpenICF Provisioner Service
- OPENIDM-18698: QueryFilter with invalid pageSize doesn’t throw an error
- OPENIDM-18738: Field Policy Service exception handler hides DS exceptions that are not policy failure exceptions
- OPENIDM-18760: Delegated admin can’t see authzMembers for internal role
- OPENIDM-18780: IDM Native console should not query audit log
- OPENIDM-18826: Out of memory in IDM platform groups read/delete members
- OPENIDM-18846: Investigate order agnostic JsonValue comparisons
- OPENIDM-18885: referencedRelationshipFields in queryConfig does not keep original data structure
- OPENIDM-18891: IDM console cli.sh throws a java.lang.NoSuchFieldError

- OPENIDM-18925: java.lang.IllegalArgumentException: Bad base context
- OPENIDM-18941: Salesforce provisioner file is overwritten when connector is enabled
- OPENIDM-19056: DS index required on reconprogressstate recon_id
- OPENIDM-19061: "Persists association" option when not selected throws "Not found error"
- OPENIDM-19181: Merry-go-round will cause duplicate RDVP calculation for signals received across conditional relationship fields
- OPENIDM-19217: Make non-returnByDefault relationship attributes available in onUpdate
- OPENIDM-19306: JDBC explicit table managed user PATCH with _fields=*_ref caused 400 error
- OPENIDM-19473: Incorrect column name "messagedetail" in VIEW being created
- OPENIDM-19492: Query for clustered recon target ids should be paged with a very small page size (e.g. 2)
- OPENIDM-19493: Conditional grantee processing speciously triggering processing of relationship fields in MOS#update
- OPENIDM-19573: Invalid and non existing cookie should return Bad Request error with OpenDJ repo
- OPENIDM-19801: Boolean attribute shows incorrect value in IDM Admin UI Level in Forgeops based deployments
- OPENIDM-20793: validateProperty appears to validate against the spelling of the properties itself

Documentation

<table>
<tr><th>Date</th><th>Description</th></tr>
<tr><td>2025-05-28</td><td>Initial release of Identity Management 7.3.2 software.</td></tr>
<tr><td>2024-04-18</td><td>Initial release of Identity Management 7.3.1 software.</td></tr>
<tr><td>2024-04-01</td><td>Added deprecation for "Sign In with LinkedIn". Refer to Deprecation → Social authentication.</td></tr>
<tr><td>2023-11-11</td><td>Added support for ForgeRock Directory Services (DS) 7.4.</td></tr>
<tr><td>2023-09-26</td><td>Updated all uses of <code>factoryPid</code> to <code>instanceName</code> and added further clarifications to the remote proxy documentation.</td></tr>
<tr><td>2023-09-14</td><td>Updated the following DS 7.3 commands:
<table>
<tr><th>OLD</th><th>NEW</th></tr>
<tr><td><code>--keyStorePasswordFile</code></td><td><code>--keyStorePassword:file</code></td></tr>
<tr><td><code>--trustStorePasswordFile</code></td><td><code>--trustStorePassword:file</code></td></tr>
</table>
These commands are used in Samples and Synchronize passwords with DS.</td></tr>
</table>

Date	Description
2023-07-27	• Added support for Red Hat Enterprise Linux 7.9. • The AD Password Plugin supports Windows Server 2022. • Added a note regarding repository upgrade.
2023-04-05	Initial release of Identity Management 7.3.0 software.

Release levels and interface stability

ForgeRock product release levels

ForgeRock defines Major, Minor, Maintenance, and Patch product release levels. The release level is reflected in the version number. The release level tells you what sort of compatibility changes to expect.

Release Level Definitions

Release Label	Version Numbers	Characteristics
Major	Version: x[.0.0] (trailing 0s are optional)	• Bring major new features, minor features, and bug fixes • Can include changes even to Stable interfaces • Can remove previously Deprecated functionality, and in rare cases remove Evolving functionality that has not been explicitly Deprecated • Include changes present in previous Minor and Maintenance releases
Minor	Version: x.y[.0] (trailing 0s are optional)	• Bring minor features, and bug fixes • Can include backwards-compatible changes to Stable interfaces in the same Major release, and incompatible changes to Evolving interfaces • Can remove previously Deprecated functionality • Include changes present in previous Minor and Maintenance releases

Release Label	Version Numbers	Characteristics
Maintenance, Patch	Version: x.y.z[.p] The optional `.p` reflects a Patch version.	• Bring bug fixes • Are intended to be fully compatible with previous versions from the same Minor release

ForgeRock product stability labels

ForgeRock products support many features, protocols, APIs, GUIs, and command-line interfaces. Some of these are standard and very stable. Others offer new functionality that is continuing to evolve.

ForgeRock acknowledges that you invest in these features and interfaces, and therefore must know when and how ForgeRock expects them to change. For that reason, ForgeRock defines stability labels and uses these definitions in ForgeRock products.

ForgeRock Stability Label Definitions

Stability Label	Definition
Stable	This documented feature or interface is expected to undergo backwards-compatible changes only for major releases. Changes may be announced at least one minor release before they take effect.
Evolving	This documented feature or interface is continuing to evolve and so is expected to change, potentially in backwards-incompatible ways even in a minor release. Changes are documented at the time of product release. While new protocols and APIs are still in the process of standardization, they are Evolving. This applies for example to recent Internet-Draft implementations, and also to newly developed functionality.
Legacy	This feature or interface has been replaced with an improved version, and is no longer receiving development effort from ForgeRock. You should migrate to the newer version, however the existing functionality will remain. Legacy features or interfaces will be marked as *Deprecated* if they are scheduled to be removed from the product.
Deprecated	This feature or interface is deprecated and likely to be removed in a future release. For previously stable features or interfaces, the change was likely announced in a previous release. Deprecated features or interfaces will be removed from ForgeRock products.
Removed	This feature or interface was deprecated in a previous release and has now been removed from the product.

Stability Label	Definition
Technology Preview	Technology previews provide access to new features that are considered as new technology that is not yet supported. Technology preview features may be functionally incomplete and the function as implemented is subject to change without notice. DO NOT DEPLOY A TECHNOLOGY PREVIEW INTO A PRODUCTION ENVIRONMENT. Customers are encouraged to test drive the technology preview features in a non-production environment and are welcome to make comments and suggestions about the features in the associated forums. ForgeRock does not guarantee that a technology preview feature will be present in future releases, the final complete version of the feature is liable to change between preview and the final version. Once a technology preview moves into the completed version, said feature will become part of the ForgeRock platform. Technology previews are provided on an “AS-IS” basis for evaluation purposes only and ForgeRock accepts no liability or obligations for the use thereof.
Internal/Undocumented	Internal and undocumented features or interfaces can change without notice. If you depend on one of these features or interfaces, contact ForgeRock support or email info@forgerock.com to discuss your needs.

Getting started

> *Guide to installing and evaluating ForgeRock® Identity Management software. This software offers flexible services for automating management of the identity life cycle.*

This guide shows you how to install and get started with ForgeRock Identity Management software. As you read this guide, you will learn how ForgeRock Identity Management software reconciles customer identity data to ensure accurate information across disparate resources within an organization.

Quick Start

Start Here

Learn about what IDM does and how it can help your organization.

Demo: Getting Started

Set up the server; understand how IDM reconciles identity data.

Where To Go from Here

Find pointers to the IDM product documentation to learn more about IDM.

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

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

About IDM

Whenever you need access to important information, administrators need to know who you are. They need to know your identity, which may be distributed in multiple accounts.

As a user, you might have several accounts even within your own company, for functions such as:

- Email
- Human Resources
- Payroll
- Engineering, Support, Accounting, and other functions

Each of these accounts may be stored in different resources, such as DS, Active Directory, OpenLDAP, and more. Keeping track of user identities in each of these resources (also known as data stores) can get complex. IDM simplifies the process, as it reconciles differences between resources.

With situational policies, IDM can handle discrepancies such as a missing or updated address for a specific user. The server includes default but configurable policies to handle such conditions. In this way, consistency and predictability is ensured, in an otherwise chaotic resource environment.

IDM can make it easier to track user identities across these resources. IDM has a highly scalable, modular, readily deployable architecture that can help you manage workflows and user information.

What Can You Do With IDM?

This software allows you to simplify the management of identity, as it can help you synchronize data across multiple resources. Each organization can maintain control of accounts within their respective domains.

IDM works equally well with user, group, and device identities.

You can also configure workflows to help users manage how they sign up for accounts, as part of how IDM manages the life cycle of users and their accounts.

You can manage employee identities as they move from job to job. You will make their lives easier as their user accounts can be registered on different systems automatically. Later, IDM can increase productivity when it reconciles information from different accounts, saving users the hassle of entering the same information on different systems.

ForgeRock Identity Management Integrations

Now that you have seen how IDM can help you manage users, review the features that IDM can bring to your organization:

- *Web-Based Administrative User Interface*

  Configure IDM with the Web-Based Administrative User Interface. You can configure many major server components without ever touching a text configuration file.

- *Self-Service Functionality*

  User self-service features can streamline onboarding, account certification, new user registration, username recovery, and password reset. The self-service features are built upon a BPMN 2.0-compliant workflow engine.

- *Registration With Social Identities*

  Users can now register new accounts using information from social identity providers, including Google, Facebook, and LinkedIn. If you configure access through more than one social identity provider, users can select and manage the providers they use. You can also synchronize user information with marketing databases.

For more information, refer to Social registration.

- *Role-Based Provisioning*

  Create and manage users based on attributes such as organizational need, job function, and geographic location.

- *Backend Flexibility*

  Choose the desired backend database for your deployment. IDM supports MySQL, Microsoft SQL Server, Oracle Database, IBM DB2, and PostgreSQL. For the supported versions of each database, refer to Before you install.

- *Password Management*

  Set up fine-grained control of passwords to ensure consistent password policies across all applications and data stores. Supports separate passwords per external resource.

- *Logging, Auditing, and Reporting*

  IDM logs all activity, internally and within connected systems. With such logs, you can track information for access, activity, authentication, configuration, reconciliation, and synchronization.

- *Access to External Resources*

  IDM can access a generic scripted connector that allows you to set up communications with many external data stores.

IDM demo : Getting started

In this guide, you will learn how IDM reconciles user data between two data stores. We will look at a department that is adding a third engineer, Jane Sanchez.

Your Human Resources department has updated their data store with Jane Sanchez's information. You want to use IDM to update the internal Engineering data store, but first, you have to start IDM.

While the reconciliation demonstrated in this guide uses two simplified data files, you can set up the same operations at an enterprise level on a variety of resources.

Return to the situation described earlier, where you have Jane Sanchez joining the engineering department. The following illustration depicts what must be done to reconcile the differences.

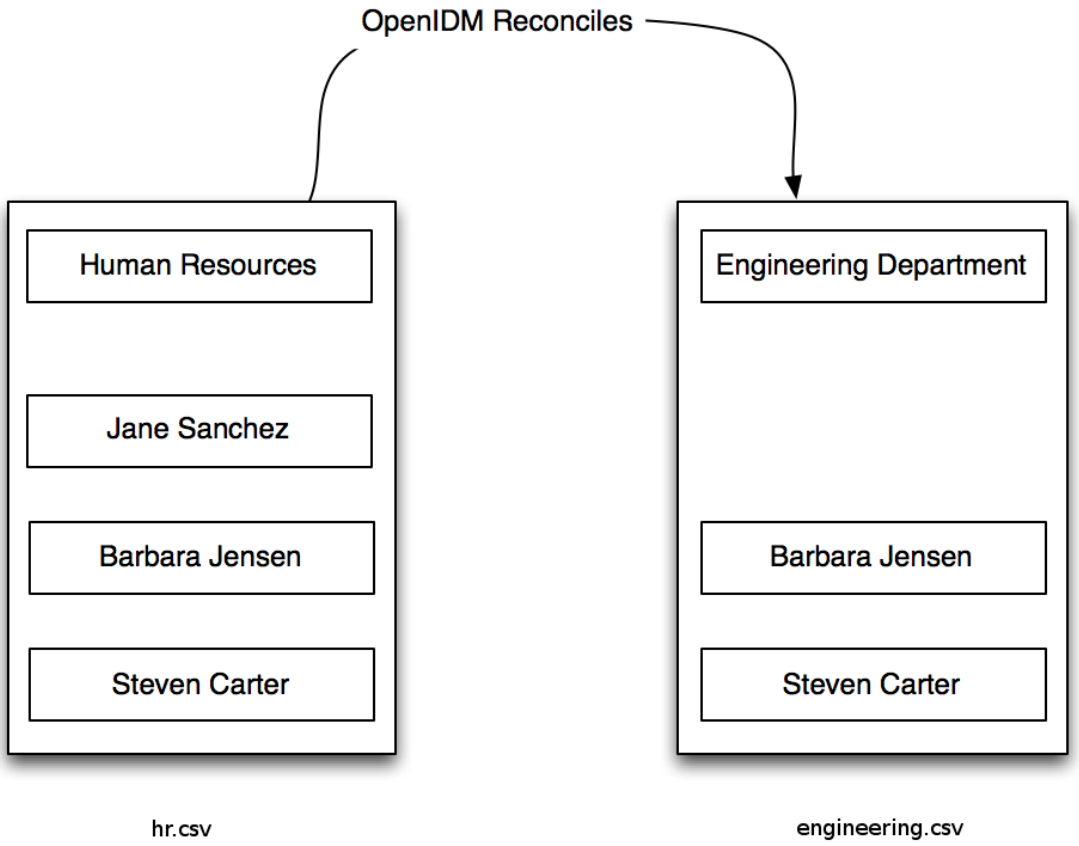


Figure 1. Reconcile Data Stores

Set up the server

What You Need Before Starting

1. For an up-to-date list of requirements, refer to Before you install.
2. Check Your Java Installation.

Download and start the server

This procedure assumes that you are starting IDM as a regular (not administrative) user named `user`.

1. Download IDM from Backstage. Releases on Backstage are thoroughly validated for ForgeRock customers who run the software in production deployments, and for those who want to try or test a given release.
2. Extract the contents of the IDM binary file to your user's `Downloads` directory. The process should unpack the contents to the `Downloads/openidm` subdirectory.
3. Navigate to the `Downloads/openidm` subdirectory:
    1. In Microsoft Windows, use Windows Explorer to navigate to the `C:\Users\user\Downloads\openidm` directory.

Double-click the `getting-started(.bat)` file. Do not select the `getting-started.sh` file, as that is intended for use on UNIX/Linux systems.

2. In Linux/UNIX, open a command-line interface and run the following command:

```
/home/user/Downloads/openidm/getting-started.sh
```

4. The following message should display:

```
-> OpenIDM ready
```

When the server is ready, you can administer it from a web browser. To do so, navigate to `http://localhost:8080/admin` or `https://localhost:8443/admin`. If you have installed the server on a remote system, substitute that hostname or IP.

> **Note**
>
> In production, you should connect to IDM via a secure port and import a CA-signed certificate into the truststore, as discussed in the Security.
> Until you install that certificate, a warning displays in your browser the first time you access IDM over a secure port.

The default username and password for the IDM Administrator is `openidm-admin` and `openidm-admin`.

When you log in to IDM at a URL with the `/admin` endpoint, you are logging into the Administrative User Interface, also known as the admin UI.

> **Warning**
>
> The default password for the administrative user, `openidm-admin`, is `openidm-admin`. To protect your deployment in production, change this password.

Demo data files

In a production deployment, you can have any number of external data stores, such as Active Directory and ForgeRock Directory Services (DS). For illustration purposes, this guide uses two simple static files as external data stores:

- `hr.csv` represents the Human Resources data store. It is in CSV format, commonly used to share data between spreadsheet applications.
- `engineering.csv` represents the Engineering data store. It is also in CSV format.

You can find these files in the binary package that you downloaded earlier, in the following subdirectory: `openidm/samples/getting-started/data`.

Reconcile data stores

A central feature of IDM is reconciliation — comparing the contents of two data stores and deciding what to do, depending on the differences.

This scenario is based on two data files:

- `hr.csv` , which represents the Human Resources data store
- `engineering.csv` , which represents the Engineering data store

Reconciliation modifies the Engineering data store by adding the newly hired Jane Sanchez. As suggested by the following illustration, it will also address detailed differences between Jane's Human Resources account and the Engineering data store.

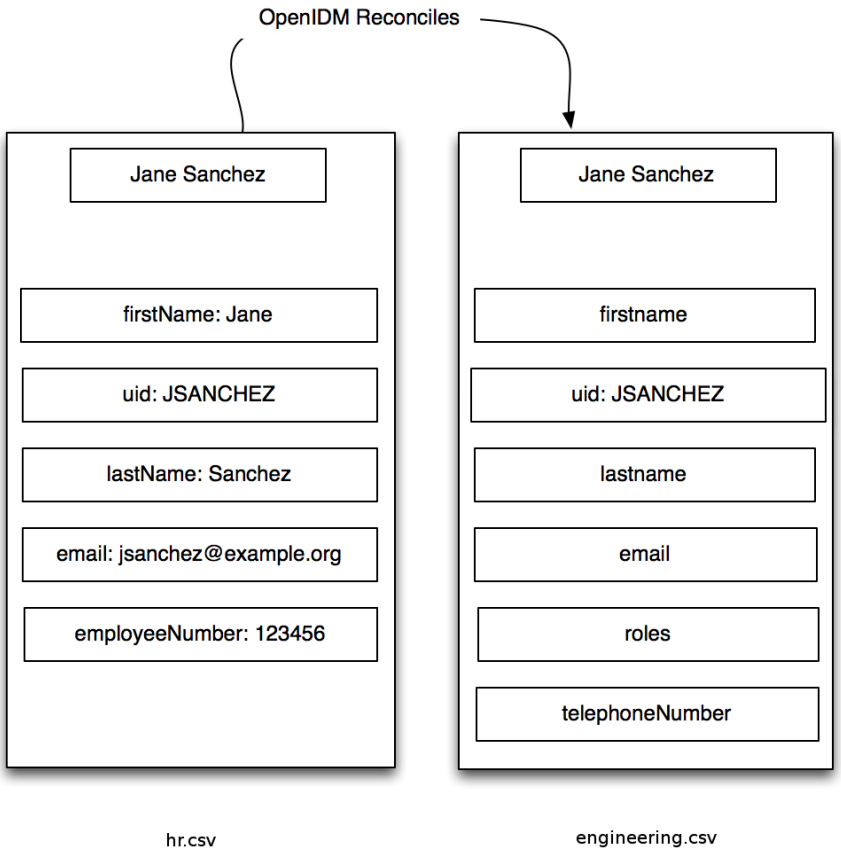


Figure 1. Data Stores Can Have Different Categories of Data

This sample includes configuration files that map detailed information from the Human Resources data store to the Engineering data store. For example, the configuration maps the `firstName` entry in Human Resources to the `firstname` entry in Engineering.

> **Note**
>
> Mapping between data stores may require additional configuration. You should find two `provisioner.openicf-*.json` files in the `/path/to/openidm/samples/getting-started/conf` subdirectory. The provisioner files configure connections to external resources, such as Active Directory, ForgeRock Directory Services (DS) or even the `engineering.csv` and `hr.csv` files used in this guide. For more information, refer to Connector reference overview.

In the admin UI, you can review how the different categories are reconciled for user Jane Sanchez. Log in to the admin UI at `https://localhost:8443/admin`. The default username is `openidm-admin` and default password is `openidm-admin`.

Click **Configure** > **Mappings** > `HumanResources_Engineering` > **Edit**.

In the *Sample source preview* text box, enter `Sanchez`. A selectable drop-down entry for Jane Sanchez should display. When you select Jane Sanchez's entry, two tables of attributes should display. One table displays how the data is modeled in the source data store. The other table displays how the data is modeled in the target data store. Refer to the following screenshot for an example:

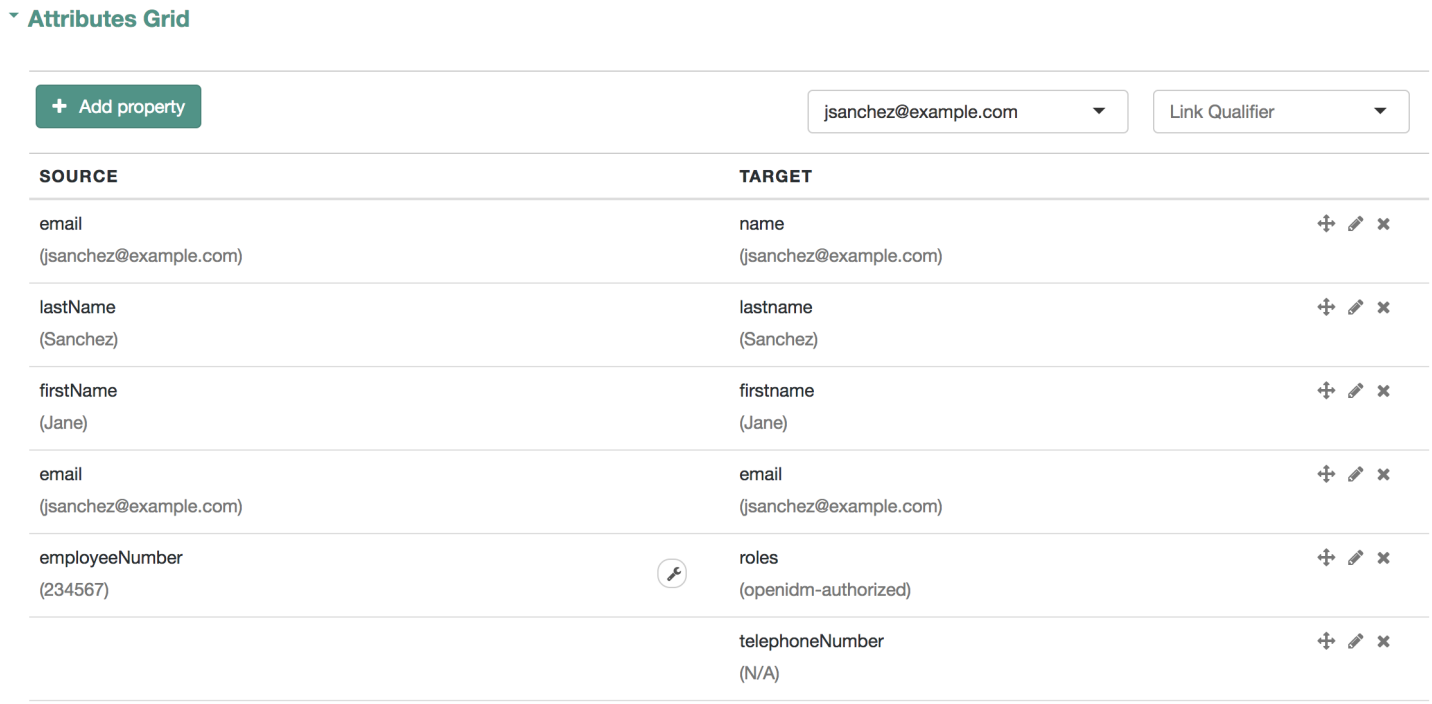


Figure 2. Reconciling Differences for an Account

Scroll back up the same page. Click **Reconcile**.

When you reconcile the two data stores, IDM makes the change to the Engineering data store.

The mapping for this example is configured in the `sync.json` file, in the `/path/to/openidm/samples/getting-started/conf` directory.

Reconcile after an update

Now that you have used IDM to reconcile two data stores, try something else. Assume the Engineering organization wants to overwrite all user telephone numbers in its employee data store with one central telephone number.

For this purpose, you can set up a default telephone number for the next reconciliation:

1. Click **Configure** > **Mappings** > `HumanResources_Engineering` > **Edit**.
2. On the **HumanResources_Engineering** mapping page, click the **Properties** tab, and expand the **Attributes** grid.
3. In the **TARGET** column, select the row that contains the `telephoneNumber` attribute.
4. Click the **Default Values** tab, and type a default number:

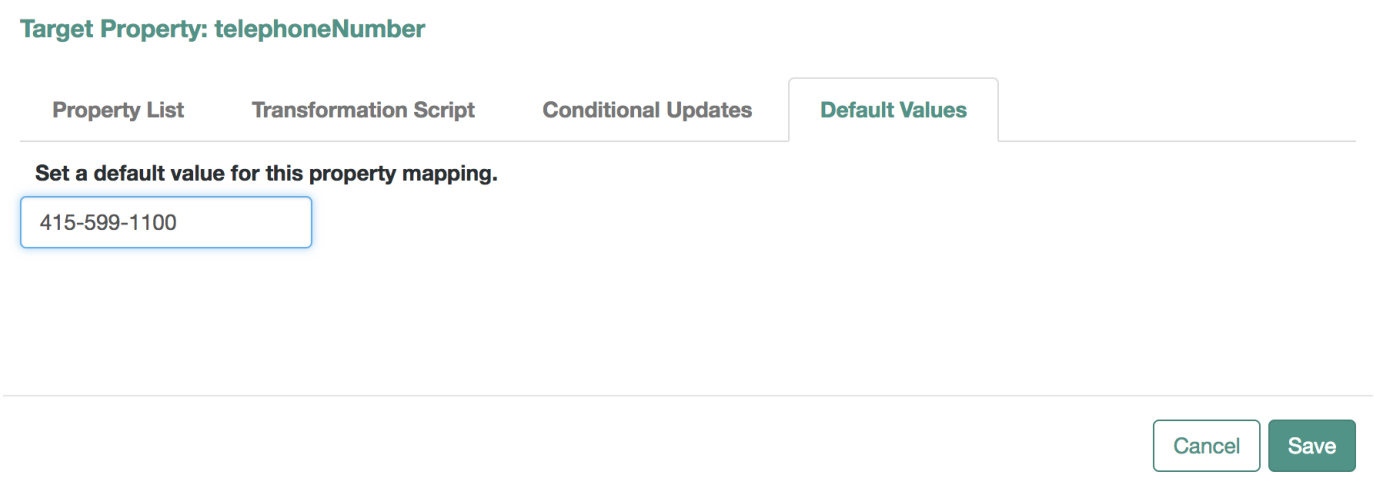


Figure 1. Set a new default telephone number

When you click **Update**, and **Save**, IDM changes the mapping in the `sync.json` file. The next time you run a reconciliation from Human Resources to Engineering, the default telephone number will be included for all employees in the Engineering group.

Stop and remove the server

Follow these steps to stop and remove IDM.

1. To stop IDM, return to the console window where you saw the following message:

   ```
   -> OpenIDM ready
   ```

   Press `Return`, and type the following command:

   ```
   -> shutdown
   ```

2. IDM is self-contained. After you shut down the server, you can choose to delete the files in the `/path/to/openidm` directory. There are no artifacts in system registries or elsewhere.

We hope that you want to continue exploring IDM.

To do so, review the rest of the IDM documentation.

Where to go from here

IDM can do much more than reconcile data between two different sources. Read about the key product features in these sections:

Reconciliation

IDM supports reconciliation between two data stores, as a source and a target.

In identity management, reconciliation compares the contents of objects in different data stores, and makes decisions based on configurable policies.

For example, if you have an application that maintains its own user store, IDM can ensure your canonical directory attributes are kept up to date by reconciling their values as they are changed.

For more information, refer to Synchronization overview.

Authentication Modules

IDM provides several authentication modules to help you protect your systems. For more information, refer to Authentication and session modules.

Password Management

Administrative users can manage user passwords from the admin UI, and users can reset their own passwords in the End User UI.

To access the End User UI as an administrative user, log in to the admin UI, and select **Self-Service** from the drop-down menu in the top right corner:

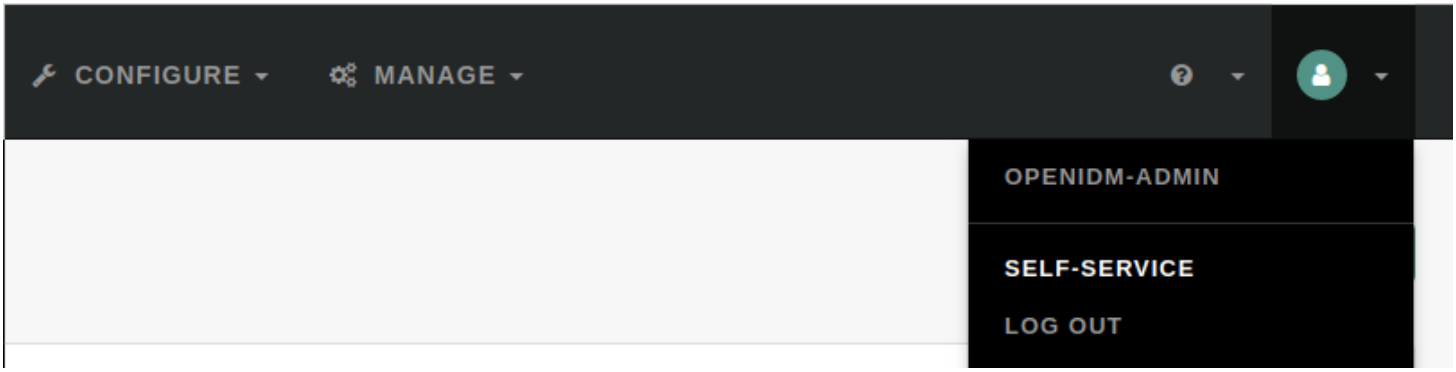


Figure 1. Access the Self-Service User Interface

In the End User UI, click **Edit Your Profile**, and click **Reset** next to the **Password** field. You can change your password, subject to the following minimum number of characters:

- Length ≥ 8
- Capital letters ≥ 1
- Numbers ≥ 1

IDM supports robust password policies. You can modify policies such as the following:

- Elements that should not be a part of a password, such as a family name
- Password expiration dates
- Password histories, to prevent password reuse

For more information, including details on configuring these policies, refer to Passwords.

User Role Management

Some users need accounts on multiple systems. For example, insurance agents may also have insurance policies with the company that they work for. In that situation, the insurance agent is also a customer of the company.

Alternatively, a salesperson may also test customer engineering scenarios. That salesperson may also need access to engineering systems.

Each of these user scenarios is known as a *role*. You can set up a consolidated set of attributes associated with each role. To do so, you would configure custom roles to assign to selected users. For example, you may assign both *insured* and *agent* roles to an agent, while assigning the *insured* role to all customers.

In a similar fashion, you can assign both *sales* and *engineering* roles to the sales engineer.

You can then synchronize users with those roles into appropriate data stores.

For more information, refer to Managed Roles. For a sample of how you can configure external roles, refer to Provision users with roles.

Business Processes and Workflows

A business process begins with an objective and includes a well-defined sequence of tasks to meet that objective. IDM allows you to configure many of these tasks as self-service workflows, such as self-registration, new user onboarding, and account certification.

You can also automate many of these tasks as a workflow.

After you configure the right workflows, a newly hired engineer can log in to IDM and request access to manufacturing information.

That request is sent to the appropriate manager for approval. After it is approved, IDM provisions the new engineer with access to manufacturing.

IDM supports workflow-driven provisioning activities, based on the embedded Flowable Process Engine, which complies with the Business Process Model and Notation 2.0 (BPMN 2.0) standard.

Remote Data Stores

IDM can connect to a substantial variety of user and device data stores, on premise and in the cloud. A number of specific connectors are provided, allowing you to connect to those dedicated data stores. In addition, you can connect to many more data stores using a scripted connector framework.

Connectors are provided for a number of external resources, including:

- Google Web Applications (refer to Google Apps connector).
- Salesforce (refer to Salesforce connector).
- Any LDAPv3-compliant directory, including DS and Active Directory (refer to LDAP connector).
- CSV Files (refer to CSV file connector).
- Database Tables (refer to Database table connector).

For a full list, refer to Supported connectors.

If the resource that you need is not on the list, you should be able to use one of the scripted connectors to connect to that resource:

- For connectors associated with Microsoft Windows, IDM includes a PowerShell Connector Toolkit that you can use to provision a variety of Microsoft services, including but not limited to Active Directory, SQL Server, Microsoft Exchange, SharePoint, Azure Active Directory, and Office 365. For more information, refer to Powershell connector. IDM includes a sample PowerShell Connector configuration, described in Connect to Active Directory with the PowerShell connector.
- For other external resources, IDM includes a Groovy Connector Toolkit that allows you to run Groovy scripts to interact with any external resource. For more information, refer to Groovy Connector Toolkit.

  For sample implementations of the scripted Groovy connector, refer to Connect to DS with ScriptedREST.

Additional Samples

IDM is a lightweight and highly customizable identity management product.

The documentation includes a number of additional use cases. Most of these are known as *Samples*, and are described in Samples provided with IDM.

These samples include step-by-step instructions on how you can connect to different data stores, customize product behavior using JavaScript and Groovy, and administer IDM with ForgeRock's common REST API commands.

Samples

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

These samples demonstrate the core functionality of ForgeRock Identity Management software. The samples correspond to the configurations provided in the `openidm/samples` directory. They cover a number of ForgeRock Identity Management features, often including multiple features in a single sample.

Important

Samples are provided as a starting point, using default configuration where appropriate. They will most likely need further customization to address the specific requirements of your deployment.

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

Samples Summary

Get a summary of all the samples in this guide.

Start Here

Setup that applies to all samples.

Sync With CSV

A basic sample showing one-way synchronization from a CSV file to IDM.

Sync With LDAP

A basic sample showing one-way synchronization from an LDAP directory to IDM.

Sync Two External Resources

Synchronize two resources without storing data in IDM.

liveSync

liveSync between external LDAP resources.

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

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

Samples provided with IDM

This section lists the samples provided with IDM (in the `openidm/samples` directory), with a high-level overview of each sample.

Synchronize Data From a CSV File to IDM

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

One-Way Synchronization From LDAP to IDM

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

Two-Way Synchronization Between LDAP and IDM

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

Synchronize LDAP Groups

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

Synchronize LDAP Group Membership

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

Synchronize Data Between Two External Resources

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

Asynchronous Reconciliation Using Workflow

The `sync-asynchronous` sample shows how you can use workflows to launch an asynchronous reconciliation operation.

LiveSync With an LDAP Server

The `livesync-with-ad` sample shows the liveSync mechanism that pushes changes from an external resource to the IDM repository. The sample uses an LDAP connector to connect to an LDAP directory, either ForgeRock Directory Services (DS) or Active Directory.

Synchronize Accounts With the Google Apps Connector

The `sync-with-google` sample uses the Google Apps Connector to create users and groups on an external Google system, and to reconcile those accounts with the IDM managed user repository.

Synchronize Users Between Salesforce and IDM

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

Synchronize Kerberos User Principals

The `sync-with-kerberos` sample demonstrates how to use the scripted Kerberos connector to manage Kerberos user principals and to reconcile user principals with IDM managed user objects.

Store Multiple Passwords For Managed Users

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

Link Multiple Accounts to a Single Identity

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

Link Historical Accounts

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

Connect to DS With ScriptedREST

The `scripted-rest-with-dj` sample uses the Groovy Connector Toolkit to implement a ScriptedREST connector that interacts with the DS REST API.

Connect to MySQL With ScriptedSQL

The `scripted-sql-with-mysql` sample uses the Groovy Connector Toolkit to implement a ScriptedSQL connector that interacts with an external MySQL database.

Synchronize Users Between IDM and AzureAD

The `sync-with-azuread` sample uses the MS Graph API connector to synchronize users between IDM and Azure AD.

Connect to Active Directory With the PowerShell Connector

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

Provision Users With Roles

The `provisioning-with-roles` sample builds on the sample described in One-way synchronization from LDAP to IDM, and demonstrates how attributes are provisioned to an external system (an LDAP directory), based on role membership.

Provision Users With Workflow

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

Direct Audit Information To MySQL

The `audit-jdbc` sample uses a ScriptedSQL implementation of the Groovy Connector Toolkit to direct audit information to a MySQL database.

Direct Audit Information to a JMS Broker

The `audit-jms` sample demonstrates how the JMS audit event handler can publish messages that comply with the Java™ Message Service Specification Final Release 1.1.

Synchronize Data Between MongoDB and IDM

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

Synchronize Data Between HubSpot and IDM

The `sync-with-hubspot` sample demonstrates bidirectional synchronization between IDM managed users and HubSpot contacts.

Synchronize Data Between DocuSign and IDM

The `sync-with-docusign` sample demonstrates bidirectional synchronization between IDM managed users and DocuSign user accounts.

Synchronize Data Between a SCIM Provider and IDM

The `sync-with-scim` sample demonstrates bidirectional synchronization between IDM managed users and roles with corresponding users and roles from a SCIM provider.

Subscribe to JMS Messages

The `scripted-jms-subscriber` sample demonstrates the scripted JMS message handler, and how it performs ForgeRock REST operations.

Authenticate Using a Trusted Servlet Filter

The `trusted-servlet-filter` sample shows how to use a custom servlet filter and the Trusted Request Attribute authentication module to let IDM authenticate through another service.

Create a Custom Endpoint

IDM supports scriptable custom endpoints that let you launch arbitrary scripts through an IDM REST URI. The `example-configurations/custom-endpoint` sample shows how custom endpoints are configured and returns a list of variables available to each method used in a custom endpoint script.

Start here

Before you try any of the samples read Run the Samples and Prepare IDM. For any samples that require an LDAP server, refer to LDAP Server Configuration.

Run the samples

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

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

When you move from one sample to the next, you are changing the IDM configuration. For more information, refer to Configuration changes.

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

Prepare IDM

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

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

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

LDAP server configuration

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

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

  Servers with replication ports maintain a changelog for their own use. The changelog is exposed over LDAP under the base DN, `cn=changelog`. For samples that demonstrate liveSync with an LDAP server, you *must* configure a replication port when you set up DS. For ease of use, all the LDAP samples assume that you have configured a replication port, even if you don't use liveSync.
- A user with DN `uid=admin` and password `password` has read access to the LDAP server.
- Directory data for that server is stored under base DN `dc=com`.
- User objects for that server are stored under base DN `ou=People,dc=example,dc=com`.
- User objects have the object class `inetOrgPerson`.
- User objects have the following attributes:
  - `cn`

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

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

Note

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

1. Shutdown DS:

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

2. Delete the `opendj/db` directory:

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

3. Delete the `opendj/config` directory:

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

Start DS using sample LDIF data

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

> **Note**
>
> The following procedure provides setup instructions for DS 7.3. For older versions of DS, or an alternative LDAP server, modify the instructions, as necessary.

1. Download the DS and IDM .zip archives.
2. Extract the .zip archives.
3. Generate a DS *deploymentId* for DS setup and deployment management:

```
/path/to/opendj/bin/dskeymgr create-deployment-id --deploymentIdPassword password
your-deployment-ID
```

4. Start DS:

```
/path/to/opendj/setup \
--serverId evaluation-only \
--deploymentId your-deployment-ID \
--deploymentIdPassword password \
--rootUserDN uid=admin \
--rootUserPassword password \
--hostname localhost \
--adminConnectorPort 4444 \
--ldapPort 1389 \
--enableStartTls \
--ldapsPort 1636 \
--replicationPort 8989 \
--httpPort 8090 \
--profile ds-user-data:7.0.0 \
--set ds-user-data/baseDn:dc=com \
--set ds-user-data/ldifFile:/path/to/openidm/samples/sync-with-ldap/data/Example.ldif \
--acceptLicense \
--start
<License Agreement>...

Validating parameters..... Done
Configuring certificates..... Done
Configuring server..... Done
Configuring profile DS user data store......... Done
Starting directory server.............. Done

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

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

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

```
/path/to/opendj/bin/dskeymgr \
export-ca-cert \
--deploymentId your-deployment-ID \
--deploymentIdPassword password \
--alias dscert \
--keyStoreFile /path/to/openidm/security/truststore \
--keyStorePassword:file /path/to/openidm/security/storepass
```

> **Note**
>
> Because each new deployment of DS has a unique deploymentId, the same certificate does not work from one sample to the next. To handle this scenario, do one of the following:
>
> - Give each subsequent sample certificate a unique alias. For example:
>   - `--alias dscert1`
>   - `--alias dscert2`
>   - `--alias dscert3`
> - Delete the old certificate from the truststore:
>
> ```
> keytool \
> -delete \
> -keystore /path/to/openidm/security/truststore \
> -alias dscert
> ```

Synchronize data from a CSV file to IDM

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

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

Sample overview

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

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

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

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

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

Sample configuration files

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

The following configuration files play important roles in this sample:

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

This file provides the configuration for this instance of the CSV connector. It describes, among other things, the connector version, the location of the CSV file resource, and the object types that are supported for this connection. For a complete understanding of connector configuration files, refer to Configure connectors.

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

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

```
{
    "mappings": [
        {
            "name" : "systemCsvfileAccounts_managedUser",
            "source" : "system/csvfile/account",
            "target": "managed/user",
            "correlationQuery": {
                "type": "text/javascript",
                "source": "var query = {'_queryId' : 'for-userName',
                    'uid' :  source.name};query;"
            },
            "properties": [
                {
                    "source": "email",
                    "target": "mail"
                },
                {
                    "source": "firstname",
                    "target": "givenName"
                },
                {
                    "source": "lastname",
                    "target": "sn"
                },
                {
                    "source": "description",
                    "target": "description"
                },
                {
                    "source": "_id",
                    "target": "_id"
                },
                {
                    "source": "name",
                    "target": "userName"
                },
                {
                    "source": "password",
                    "target": "password"
                },
                {
                    "source" : "mobileTelephoneNumber",
                    "target" : "telephoneNumber"
                },
                {
                    "source" : "roles",
                    "transform" : {
                        "type" : "text/javascript",
                        "source" : "var _ = require('lib/lodash'); _.map(source.split(','),
                         function(role) { return {'_ref': 'internal/role/' + role} });"
                    },
                    "target" : "authzRoles"
                }
            ],
            "policies": [
                {
                    "situation": "CONFIRMED",
                    "action": "UPDATE"
                },
                {
```

```
                    "situation": "FOUND",
                    "action": "IGNORE"
                },
                {
                    "situation": "ABSENT",
                    "action": "CREATE"
                },
                {
                    "situation": "AMBIGUOUS",
                    "action": "IGNORE"
                },
                {
                    "situation": "MISSING",
                    "action": "IGNORE"
                },
                {
                    "situation": "SOURCE_MISSING",
                    "action": "IGNORE"
                },
                {
                    "situation": "UNQUALIFIED",
                    "action": "IGNORE"
                },
                {
                    "situation": "UNASSIGNED",
                    "action": "IGNORE"
                }
            ]
        }
    ]
}
```

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

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

For more information about synchronization, reconciliation, and mappings, refer to Synchronization.

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

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

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

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

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

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

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

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

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

Run the sample

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

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

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

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

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

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

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

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

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

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true"
{
  "result": [
    {
      "_id": "bjensen",
      "_rev": "00000000e17186b6",
      "mail": "bjensen@example.com",
      "givenName": "Barbara",
      "sn": "Jensen",
      "description": "Created By CSV",
      "userName": "bjensen",
      "telephoneNumber": "1234567",
      "accountStatus": "active",
      "effectiveAssignments": [],
      "effectiveRoles": []
    },
    {
      "_id": "scarter",
      "_rev": "00000000970685c3",
      "mail": "scarter@example.com",
      "givenName": "Steven",
      "sn": "Carter",
      "description": "Created By CSV",
      "userName": "scarter",
      "telephoneNumber": "1234567",
      "accountStatus": "active",
      "effectiveAssignments": [],
      "effectiveRoles": []
    }
  ],
  ...
}
```

You can user any query filter to return the information you need. For more information, refer to Define and call data queries.

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

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

This command returns bjensen's complete user record.

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

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

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

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

6. If you enabled the scheduled reconciliation in Step 1, you can simply wait for the reconciliation operation to run. Otherwise, run the reconciliation manually with the same command you used in that step.

7. After the reconciliation has run, query the managed user repository to view the new user in the list of managed users:

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

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

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

For more information on reconciliation operations via REST, refer to Manage reconciliation.

You configure the action that IDM takes for each situation in the mapping file, `conf/sync.json`. For the list of all possible situations and actions, refer to Synchronization situations and actions.

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

IDM includes a browser-based Administrative User Interface, known as the admin UI.

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

You should now see the **Dashboard** screen, with *quick start cards* for common administrative tasks.

1. Reconcile the two resources as follows:

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

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

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

One-way synchronization from LDAP to IDM

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

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

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

Prepare the sample

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

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

Run the sample

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

1. Reconcile the repository:

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

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

2. Retrieve the users from the repository:

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

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

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

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

2. Select **Configure > Mappings** .

   The **Mappings** page displays one mapping, from the `ldap` server to the IDM repository (`managed/user`).

3. Select the mapping, and click **Reconcile** .

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

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

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

      IDM displays the two users.

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

      The **User** details page displays.

Two-way synchronization between LDAP and IDM

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

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

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

Prepare the sample

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

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

Run the sample

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

1. Reconcile the repository over the REST interface:

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

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

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

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

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

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

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

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

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

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

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

   Note the additional user entry.

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

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

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

   The **Mappings** page displays two configured mappings, one from the `ldap` server to the IDM repository (`managed/user`) and one from the repository to the `ldap` server.
3. Select the **LDAP to managed user** mapping, and click **Reconcile** .

   The reconciliation operation creates the two users from the LDAP server in the IDM repository.
4. To view the new users in the repository, from the navigation bar, click **Manage > User** .

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

> **Note**
>
> By default, *implicit synchronization* is enabled for mappings *from* the `managed/user` repository *to* any external resource. This means that when you update a managed object, any mappings defined in the `sync.json` file that have the managed object as the source are automatically executed to update the target system. For more information, refer to Resource mapping.

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

      IDM displays the user's mapped external resource.

Synchronize LDAP groups

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

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

Sample overview

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

`systemLdapAccounts_managedUser`

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

`managedUser_systemLdapAccounts`

Synchronizes users from the IDM repository to the LDAP server.

`systemLdapGroups_managedGroup`

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

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

Prepare the sample

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

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

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

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

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

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

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

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

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

Run the sample

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

1. Reconcile the group objects over the REST interface:

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

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

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

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

```
        "name": "openidm"
      }
    ],
    ...
}
```

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

   The **Mappings** page displays three configured mappings:

   - From the `ldap` server user accounts to the IDM repository (`managed/user`).
   - From the IDM managed users back to the `ldap` accounts.
   - From the `ldap` server group entries to the IDM `managed/group` entries.
3. Select the **LDAP groups to managed groups** mapping, and click **Reconcile** .

   The reconciliation operation creates the two groups from the LDAP server in the IDM repository.
4. From the navigation bar, click **Manage > Group** .

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

Synchronize LDAP group membership

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

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

Prepare the sample

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

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

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

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

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

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

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

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

Run the sample

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

1. Reconcile the repository over the REST interface:

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

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

2. Retrieve the user IDs from the repository:

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

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

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

Note

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

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

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '[
  {
    "operation": "replace",
    "field": "/ldapGroups",
    "value": ["cn=openidm2,ou=Groups,dc=example,dc=com"]
  }
]' \
"http://localhost:8080/openidm/managed/user/1eaca03d-aef7-415a-99d9-bfd3f442ef51?_action=patch"
{
  "displayName": "John Doe",
  "description": "Created for OpenIDM",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "1-415-599-1100",
  "sn": "Doe",
  "userName": "jdoe",
  "ldapGroups": [
    "cn=openidm2,ou=Groups,dc=example,dc=com"
  ],
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": [],
  "_rev": "00000000d0ebe04c",
  "_id": "1eaca03d-aef7-415a-99d9-bfd3f442ef51"
}
```

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

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account/?_queryFilter=uid+eq+'jdoe'"
{
  "result": [
    {
      "_id": "0da50512-79bb-3461-bd04-241ee4c785bf",
      "employeeType": [],
      "objectClass": [
        "person",
        "organizationalPerson",
        "inetOrgPerson",
        "top"
      ],
      "cn": "John Doe",
      "uid": "jdoe",
      "ldapGroups": [
        "cn=openidm2,ou=Groups,dc=example,dc=com"
      ],
      "givenName": "John",
      "mail": "jdoe@example.com",
      "aliasList": [],
      "dn": "uid=jdoe,ou=People,dc=example,dc=com",
      "sn": "Doe",
      "description": "Created for OpenIDM",
      "telephoneNumber": "1-415-599-1100",
      "kbaInfo": []
    }
  ],
  ...
}
```

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

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

   The reconciliation operation creates the two users from the LDAP server in the IDM repository.
4. To view the new users in the repository, from the navigation bar, click **Manage > User** .

   IDM displays the two new users.
5. From the **Users List** page, click the user `jdoe` .
6. Click the **Linked Systems** tab.

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

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

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

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

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

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

8. To verify the change, reload John Doe's user information:

    1. Reload John Doe's user information page.
    2. Click the **Linked Systems** tab.

       Observe the value of the `ldapGroups` property.

Synchronize data between two external resources

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

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

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

Configure email for the sample

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

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

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

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

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

2. Edit `external.email.json` for outbound email.

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

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

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

Run the sample

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

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

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

2. Examine the data files.

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

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

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

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

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

   For more information about implicit synchronization, refer to Synchronization types.

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

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

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

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

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

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

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

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

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

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

7. Rerun the reconciliation and query REST commands shown previously.

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

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

Asynchronous reconciliation using workflow

This sample demonstrates asynchronous reconciliation using workflows.

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

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

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

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

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

Run the sample

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

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

1. Edit the `/path/to/openidm/samples/sync-asynchronous/conf/datasource.jdbc-default.json` file with the details of your JDBC repository. For more information, refer to Select a repository.
2. Start IDM with the configuration for this sample:

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

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

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

4. Run reconciliation over the REST interface:

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

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

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

5. Review the approval tasks launched by the reconciliation.

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

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

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

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

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

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

6. Complete each approval task.

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

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

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

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

     For example, to approve the first request:

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

Repeat this command for each task ID.

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

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

LiveSync with an LDAP server

This sample resembles the sample described in Synchronize data between two external resources. However, this sample demonstrates *liveSync* from one external LDAP resource to another. LiveSync is the mechanism by which changes are pushed from an external resource to IDM and then, optionally, to another external resource. For more information, refer to Synchronization types.

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

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

Live AD Instance

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

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

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

Simulated AD Instance

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

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

Set up the LDAP resources

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

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

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

`host`

The hostname or IP address of the AD server.

`port`

The LDAP port; 389 by default.

`ssl`

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

`principal`

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

`credentials`

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

`baseContexts`

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

`baseContextsToSynchronize`

Set to the same value as `baseContexts`.

`accountSearchFilter`

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

`accountSynchronizationFilter`

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

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

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

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

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

2. Load the simulated data into that suffix:

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

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

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

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

Run the sample

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

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

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

Reconcile the two LDAP data stores

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

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

1. Run reconciliation over the REST interface:

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

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

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

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

2. List all the users in the DS server:

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

Your DS server should now contain four users:

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

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

Configure liveSync

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

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

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

Test liveSync

Test liveSync as follows:

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

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

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

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

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

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

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

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

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

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

Synchronize accounts with the Google Apps connector

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

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

Prepare the sample

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

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

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

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

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

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

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

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

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

5. Click **Create**.

6. On the **OAuth client created** pop-up, make a note of your Client ID and Client Secret.

4. Add IDM to the Trusted Apps list:

   1. Log in to the Google Admin Console.

   2. From the top left menu, select **Security > API Controls** .

   3. Select **MANAGE THIRD-PARTY APP ACCESS** , then click **Change Access** , and change the IDM app settings to **Trusted** .

Configure the Google Apps connector

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

1. Start IDM with the Google Apps sample configuration:

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

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

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

3. Select **Configure > Connectors**, and click the Google Apps connector.

4. On the **Details** tab, enable the connector.

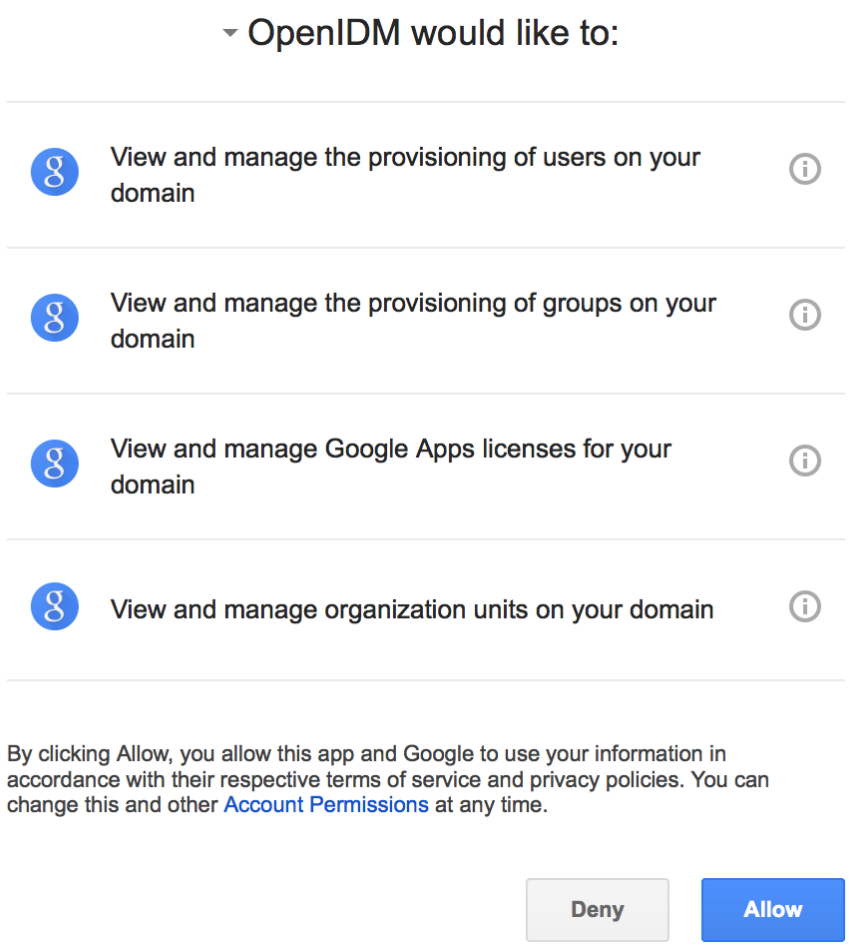
5. In the **Base Connector Details** area, enter the **Client ID** and **Client Secret** that you obtained in the previous section.

6. Click **Save**.

   IDM redirects you to a **Sign in with Google** page.

7. Log in.

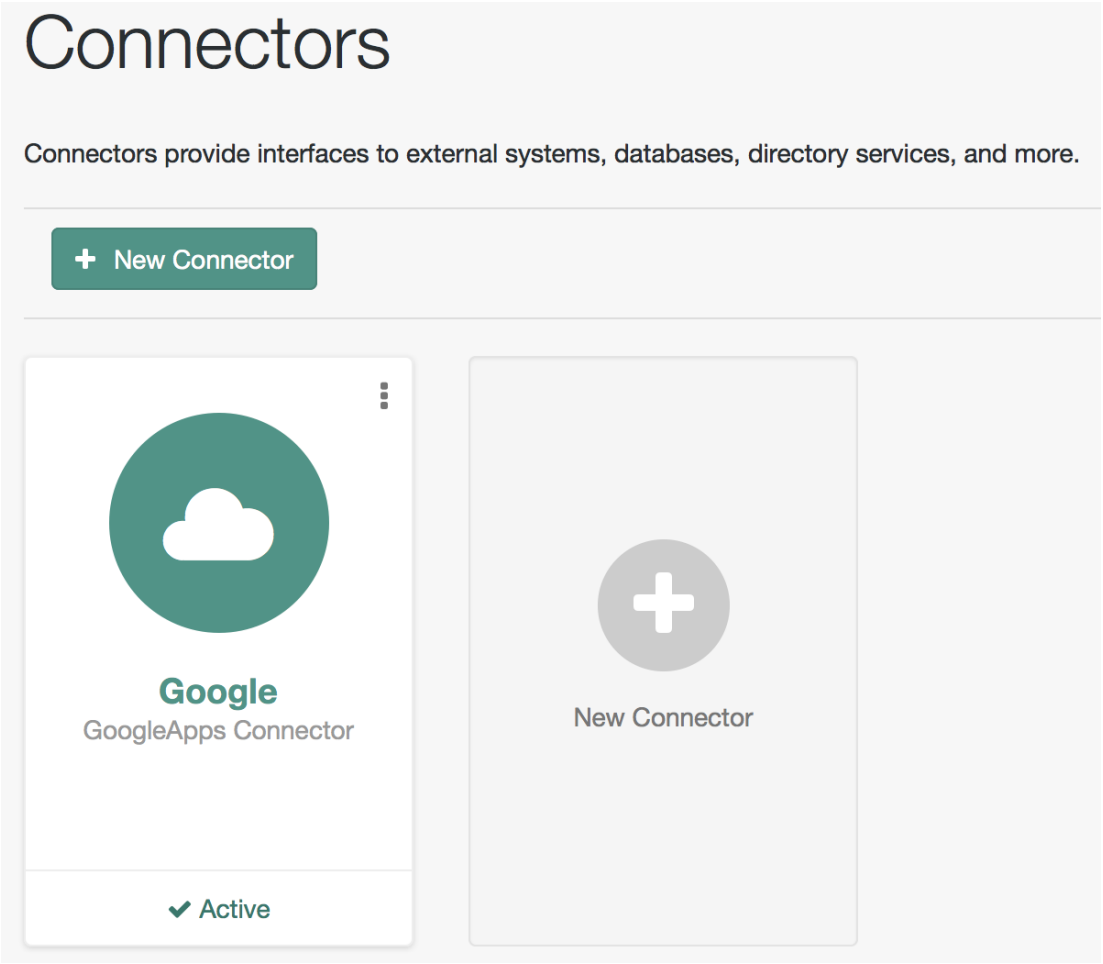
   After you have logged in, Google requests that you allow access from your project; in this case, IDM.



8. Click **Allow**.

> **Note**
>
> If you click **Deny**, you must return to the **Connector Configuration > Details** tab in the admin UI, and save your changes again.

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



Run the sample

This procedure uses create, read, update, and delete (CRUD) operations on the Google resource, to verify that the connector is working as expected. The procedure uses a combination of REST commands, to manage objects on the Google system, and the admin UI, to reconcile users from the Google system to the managed user repository.

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

The commands shown here assume that your domain is `example.com`. Adjust the examples to match your domain.

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

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

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "__NAME__": "samcarter@example.com",
  "__PASSWORD__"  : "password",
  "givenName" : "Sam",
  "familyName": "Carter",
  "agreedToTerms": true,
  "changePasswordAtNextLogin" : false
}' \
"http://localhost:8080/openidm/system/google/__ACCOUNT__?_action=create"
```

Which returns:

```
{
  "_id": "103567435255251233551",
  "_rev": "\"iwpzoDgSq9BJw-XzORg0bILYPVc/LWHPMXXG8M0cjQAPITM95Y636cM\"",
  "orgUnitPath": "/",
  "isAdmin": false,
  "fullName": "Sam Carter",
  "customerId": "C02rsqddz",
  "relations": null,
  "nonEditableAliases": null,
  "suspensionReason": null,
  "includeInGlobalAddressList": true,
  "givenName": "Sam",
  "addresses": null,
  "isDelegatedAdmin": false,
  "changePasswordAtNextLogin": false,
  "isMailboxSetup": true,
  "__NAME__": "samcarter@example.com",
  "agreedToTerms": true,
  "externalIds": null,
  "ipWhitelisted": false,
  "aliases": null,
  "lastLoginTime": [
    "1970-01-01T00:00:00.000Z"
  ],
  "organizations": null,
  "suspended": false,
  "deletionTime": null,
  "familyName": "Carter",
  "ims": null,
  "creationTime": [
    "2016-02-02T12:52:30.000Z"
  ],
  "thumbnailPhotoUrl": null,
  "emails": [
    {
      "address": "samcarter@example.com",
      "primary": true
    }
  ],
  "phones": null
}
```


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

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

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

   1. In the admin UI, select **Configure > Mappings**.
   2. Click on the **sourceGoogle__ACCOUNT___managedUser** mapping, and click **Reconcile**.
   3. Select **Manage > User** and verify that the user Sam Carter has been created in the repository.

3. Update Sam Carter's phone number in your Google resource by sending a PUT request with the updated data, and specifying the user `_id` in the request:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PUT \
--header "If-Match: *" \
--data '{
  "__NAME__": "samcarter@example.com",
  "givenName" : "Sam",
  "familyName": "Carter",
  "agreedToTerms": true,
  "changePasswordAtNextLogin" : false,
  "phones" : [
    {
      "value": "1234567890",
      "type": "home"
    },
    {
      "value": "0987654321",
      "type": "work"
    }
  ]
}' \
"http://localhost:8080/openidm/system/google/__ACCOUNT__/103567435255251233551"
{
  "_id": "103567435255251233551",
  "_rev": "\"iwpzoDgSq9BJw-XzORg0bILYPVc/vfSJgHt-STUUto4lM_4ESO9izR4\"",
  ...
  "emails": [
    {
      "address": "samcarter@example.com",
      "primary": true
    }
  ],
  "phones": [
    {
      "value": "1234567890",
      "type": "home"
    },
    {
      "value": "0987654321",
      "type": "work"
    }
  ]
}
```

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

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

5. Create a group entry on your Google resource:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "__NAME__": "testGroup@example.com",
  "__DESCRIPTION__": "Group used for google-connector sample.",
  "name": "TestGroup"
}' \
"http://localhost:8080/openidm/system/google/__GROUP__?_action=create"
{
  "_id": "00meukdy40gpg98",
  "_rev": "\"iwpzoDgSq9BJw-XzORg0bILYPVc/LLhHx2plMJPKeY1-h6eX_OVDi4c\"",
  "adminCreated": true,
  "__NAME__": "testgroup@example.com",
  "aliases": null,
  "nonEditableAliases": null,
  "__DESCRIPTION__": "Group used for google-connector sample.",
  "name": "TestGroup",
  "directMembersCount": 0
}
```

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
-H 'If-Match: "iwpzoDgSq9BJw-XzORg0bILYPVc/LLhHx2plMJPKeY1-h6eX_OVDi4c"' \
--request PUT \
--data '{
  "members": [
    {
      "role": "MEMBER",
      "email": "samcarter@example.com"
    }
  ]
}' \
"http://localhost:8080/openidm/system/google/__GROUP__/00meukdy40gpg98"
{
  "_id": "00meukdy40gpg98/samcarter@example.com",
  "_rev": "iwpzoDgSq9BJw-XzORg0bILYPVc/CPNpkRnowkGWRvNQvUK9ev6gQ90",
  "__NAME__": "00meukdy40gpg98/samcarter@example.com",
  "role": "MEMBER",
  "email": "samcarter@example.com",
  "type": "USER",
  "groupKey": "103567435255251233551"
}
```

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/google/__GROUP__/00meukdy40gpg98"
{
  "_id": "00meukdy40gpg98",
  "_rev": "iwpzoDgSq9BJw-XzORg0bILYPVc/chUdq5m5_cycV2G4sdl7ZKAF75A",
  "adminCreated": true,
  "__NAME__": "testgroup@example.com",
  "aliases": null,
  "nonEditableAliases": [
    "testGroup@example.test-google-a.com"
  ],
  "__DESCRIPTION__": "Group used for google-connector sample.",
  "name": "TestGroup",
  "directMembersCount": 1
}
```

8. Delete the group entry:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/system/google/__GROUP__/00meukdy40gpg98"
{
  "_id": "00meukdy40gpg98",
  "_rev": "iwpzoDgSq9BJw-XzORg0bILYPVc/chUdq5m5_cycV2G4sdl7ZKAF75A",
  "adminCreated": true,
  "__NAME__": "testgroup@example.com",
  "aliases": null,
  "nonEditableAliases": [
    "testGroup@example.com.test-google-a.com"
  ],
  "__DESCRIPTION__": "Group used for google-connector sample.",
  "name": "TestGroup",
  "directMembersCount": 1
}
```

The delete request returns the complete group object.

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/system/google/__ACCOUNT__/103567435255251233551"
{
  "_id": "103567435255251233551",
  "_rev": "iwpzoDgSq9BJw-XzORg0bILYPVc/ah6xBLujMAHieSWSisPa1CV6T3Q",
  "orgUnitPath": "/",
  "isAdmin": false,
  "fullName": "Sam Carter",
  "customerId": "C02rsqddz",
  "relations": null,
  "nonEditableAliases": [
    "samcarter@example.com.test-google-a.com"
  ],
  "suspensionReason": null,
  "includeInGlobalAddressList": true,
  "givenName": "Sam",
  "addresses": null,
  "isDelegatedAdmin": false,
  "changePasswordAtNextLogin": false,
  "isMailboxSetup": true,
  "__NAME__": "samcarter@example.com",
  "agreedToTerms": true,
  "externalIds": null,
  "ipWhitelisted": false,
  "aliases": null,
  "lastLoginTime": [
    "1970-01-01T00:00:00.000Z"
  ],
  "organizations": null,
  "suspended": false,
  "deletionTime": null,
  "familyName": "Carter",
  "ims": null,
  "creationTime": [
    "2016-02-02T12:52:30.000Z"
  ],
  "thumbnailPhotoUrl": null,
  "emails": [
    {
      "address": "samcarter@example.com",
      "primary": true
    }
  ],
  "phones": [
    {
      "value": "1234567890",
      "type": "home"
    },
    {
```

```
      "value": "0987654321",
      "type": "work"
    }
  ]
}
```

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

Synchronize users between Salesforce and IDM

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

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

Prepare the sample

1. Configure your Salesforce organization.

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

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

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

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

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

Run the sample

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

Use the admin UI

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

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

    1. Select the **Salesforce** connector, and click **Enable**.

    2. Under **Base Connector Details**, select **Production**, **Sandbox**, or **Custom** to set your Login URL.

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

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

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

    3. Enter your Consumer Key and Consumer Secret, then select **Save** to update the connector configuration.

    4. The connector now attempts to access your Salesforce organization.

    5. Enter your Salesforce login credentials.

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

3. To test reconciliation, select **Configure > Mappings**.

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

4. Select **Reconcile** on the first mapping.

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

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

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

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

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

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

Use the command line

This section breaks the sample into two tasks:

- Configure the connector.
- Test the configuration.

Configure the Salesforce connector

1. Retrieve all the required configuration properties, as described in Configure the Salesforce connector with a configuration file.

2. Edit the `configurationProperties` object in the Salesforce connector configuration file (`openidm/samples/sync-with-salesforce/conf/provisioner.openicf-salesforce.json`) to include your Salesforce login URL, Consumer Key and Consumer Secret, refresh token, and instance URL.

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

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

```
{
    "enabled" : true,
...
    "configurationProperties" : {
        "connectTimeout" : 120000,
        "loginUrl" : https://login.salesforce.com/services/oauth2/token,
        "idleCheckInterval" : 10000,
        "refreshToken" : "5Aep861KIwKdekr90I4iHdtDgWwRoG7O_6uHrgJ.yVtMS0UaGxRqE6WFM...",
        "clientSecret" : "4850xxxxxxxxxxxx425",
        "clientId" : "3MVG98dostKihXN7Is8Q0g5q1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxP...",
        "instanceUrl" : "https://example-com.cs1.my.salesforce.com",
        "version" : 44
    }
...
```

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=test"
{
  "name": "salesforce",
  "enabled": true,
  "config": "config/provisioner.openicf/salesforce",
  "connectorRef": {
    "bundleVersion": "1.5.20.14",
    "bundleName": "org.forgerock.openicf.connectors.salesforce-connector",
    "connectorName": "org.forgerock.openicf.connectors.salesforce.SalesforceConnector"
  },
  "displayName": "Salesforce Connector",
  "objectTypes": [
    "__ALL__",
    "User"
  ],
  "ok": true
}
```

Run reconciliation

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

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

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

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

2. Retrieve the managed users in the repository:

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

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

Synchronize Kerberos user principals

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

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

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

Configure the Kerberos connector

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

`host`

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

`port`

The SSH port on that machine.

Default: `22` (the default SSH port)

`user`

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

`password`

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

`prompt`

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

`customConfiguration`

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

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

For more information on setting this property, refer to `customConfiguration` .

`customSensitiveConfiguration`

The password for the user principal.

For more information on setting this property, refer to `customSensitiveConfiguration`.

Your connector configuration should look something like the following:

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

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

For information about the complete Kerberos connector configuration, refer to Configure the Kerberos connector.

Caution

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

Run the sample

This sample demonstrates IDM communicating with the Kerberos Server, creating/deleting users, and reconciling the IDM repository with Kerberos.

1. Start IDM with the configuration for the Kerberos sample:

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

2. Test that your connector configuration is correct and that IDM can reach your Kerberos server:

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

If the command returns `"ok": true`, your configuration is correct. Continue with the sample.

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

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

4. Create two new managed users, using REST or the admin UI.

REST

The following commands create users `bjensen` and `scarter` over REST:

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

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

admin UI

To create users `bjensen` and `scarter` using the admin UI, select **Managed > User**, and click **New User**.

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

REST

The following command creates runs the reconciliation over REST:

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

admin UI

To run the reconciliation using the admin UI, select **Configure > Mappings**, click on the `managedUser_systemKerberos` mapping, and click **Reconcile**.

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

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

7. Retrieve the `bjensen` complete user principal from the Kerberos server over REST, or using the admin UI:

REST

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

admin UI

To retrieve the user using the admin UI, select **Manage > User**, click **bjensen**, and click the **Linked Systems** tab to display the corresponding Kerberos server entry.

Note

The default values for properties such as `maximumRenewableLife` are set in your connector configuration. For more information, refer to Configure the Kerberos connector.

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

    1. First, obtain the ID by querying the `userName`:

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

2. Now delete the user with ID `ce3d9b8f-1d15-4950-82c1-f87596aadcb6` over REST, or using the admin UI. This ID will be different in your example.

REST

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

admin UI

To delete managed user `bjensen` using the admin UI, select **Manage > User**, select the checkbox adjacent to **bjensen**, and click **Delete Selected**.

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

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

10. Retrieve the list of Kerberos user principals again. The Kerberos principal for `bjensen` should not exist:

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

Note

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

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

Store multiple passwords for managed users

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

Note

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

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

Configure the multiple passwords sample

This sample assumes the following scenario:

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

  For the purposes of this sample, the two servers are represented by two separate organizational units on a single ForgeRock Directory Services (DS) instance.
- Managed user objects have two additional password fields, each mapped to one of the two LDAP servers.
- Both LDAP servers have a requirement for a password history policy, but the history size differs for the two policies.

  The sample shows how to extend the password history policy, described in Creating a Password History Policy, to apply to multiple password fields.
- The value of a managed user's `password` field is used by default for the additional passwords *unless* the CREATE, UPDATE, or PATCH requests on the managed user explicitly contain a value for these additional passwords.

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

`provisioner.openicf-ldap.json`

Configures the connection to the first LDAP directory.

`provisioner.openicf-ldap2.json`

Configures the connection to the second LDAP directory.

`sync.json`

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

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

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

`managed.json`

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

This file has been customized as follows:

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

`router.json`

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

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

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

Password history policy

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

The following configuration changes set up the password history policy:

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

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

- The `onCreate-user-custom.js` script performs the standard `onCreate` tasks for a managed user object but also stores the initial value of each of the fields for which IDM should keep a history. The script is passed the following configurable properties:

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

- The `onUpdate-user-custom.js` script compares the old and new values of the historical fields on update events to determine if the values have changed. When a new value is detected, it is stored in the list of historical values for that field.

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

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

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

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

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

LDAP server configuration

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

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

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

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

Show multiple accounts

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

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

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

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
  "mail": "john.doe@example.com",
  "password": "Secretpw1",
  "ldapPassword": "S3cretPw",
  "ldap2Password": "Secr3tpw1"
}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "_id": "5ce188f6-252b-429e-aad1-4d8754d77de5",
  "_rev": "00000000d2d76089",
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
  "mail": "john.doe@example.com",
  "ldapPassword": {
    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "cipher": "AES/CBC/PKCS5Padding",
        "stableId": "openidm-sym-default",
        "salt": "lkackh...",
        "data": "T0mljk...",
        "keySize": 16,
        "purpose": "idm.password.encryption",
        "iv": "ehSMbdNn...",
        "mac": "PssPOsW..."
      }
    }
  },
  "ldap2Password": {
    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "cipher": "AES/CBC/PKCS5Padding",
        "stableId": "openidm-sym-default",
        "salt": "lSzMTU54...",
        "data": "UWlQo5Ws...",
        "keySize": 16,
        "purpose": "idm.password.encryption",
        "iv": "ehSMbdN...",
        "mac": "PssPOs..."
      }
    }
```

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

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

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

3. As a result of implicit synchronization, two separate LDAP accounts should have been created for user jdoe on our two simulated LDAP servers. For more information about implicit synchronization, refer to Synchronization types.

4. Query the IDs in the LDAP directory as follows:

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

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

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

Note

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

```
Do you trust this server certificate?

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

Enter choice [1]: 2
```

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

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

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
  "operation": "replace",
  "field": "ldapPassword",
  "value": "TTestw0rd"
} ]' \
"http://localhost:8080/openidm/managed/user/5ce188f6-252b-429e-aad1-4d8754d77de5"
{
  "_id": "5ce188f6-252b-429e-aad1-4d8754d77de5",
  "_rev": "000000001298f6a6",
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
  ...
  "ldapPassword": {
    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "cipher": "AES/CBC/PKCS5Padding",
        "stableId": "openidm-sym-default",
        "salt": "Vlco8e...",
        "data": "INj9lk...",
        "keySize": 16,
        "purpose": "idm.password.encryption",
        "iv": "ehSMbdNn...",
        "mac": "PssPOsW..."
      }
    }
  },
  ...
}
```

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

> **Note**
>
> For the following command, make sure to enter `2` or `3` at the following prompt:
>
> ```
> Do you trust this server certificate?
>
>   1) No
>   2) Yes, for this session only
>   3) Yes, also add it to a truststore
>   4) View certificate details
>
> Enter choice [1]: 2
> ```

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

Show the password history policy

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

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
  "operation": "replace",
  "field": "ldapPassword",
  "value": "TTestw0rd1"
} ]' \
"http://localhost:8080/openidm/managed/user/5ce188f6-252b-429e-aad1-4d8754d77de5"
{
  "_id": "5ce188f6-252b-429e-aad1-4d8754d77de5",
  "_rev": "00000000a92657c7",
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
  "mail": "john.doe@example.com",
  ...
  "ldapPassword": {
    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "cipher": "AES/CBC/PKCS5Padding",
        "stableId": "openidm-sym-default",
        "salt": "TjolL7...",
        "data": "Unbalo...",
        "keySize": 16,
        "purpose": "idm.password.encryption",
        "iv": "ehSMbdNn...",
        "mac": "PssPOsW..."
      }
    }
  },
  ...
}
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
  "operation": "replace",
  "field": "ldapPassword",
  "value": "TTestw0rd2"
} ]' \
"http://localhost:8080/openidm/managed/user/5ce188f6-252b-429e-aad1-4d8754d77de5"
{
  "_id": "5ce188f6-252b-429e-aad1-4d8754d77de5",
  "_rev": "00000000dc6160c8",
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
  ...
  "ldapPassword": {
    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "cipher": "AES/CBC/PKCS5Padding",
        "stableId": "openidm-sym-default",
        "salt": "Ynio9n...",
        "data": "R0ol2b...",
        "keySize": 16,
        "purpose": "idm.password.encryption",
        "iv": "ehSMbdNn...",
        "mac": "PssPOsW..."
      }
    }
  },
  ...
}
```

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

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

2. The history size for the `ldapPassword` policy is set to 2. To demonstrate the history policy, attempt to patch jdoe's entry with a password value that was used in his previous 2 password changes: `TTestw0rd2`:

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

The password change fails the `IS_NEW` policy requirement.

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
  "operation": "replace",
  "field": "ldapPassword",
  "value": "TTestw0rd"
} ]' \
"http://localhost:8080/openidm/managed/user/5ce188f6-252b-429e-aad1-4d8754d77de5"
{
  "_id": "5ce188f6-252b-429e-aad1-4d8754d77de5",
  "_rev": "00000000792afa08",
  "userName": "jdoe",
  "givenName": "John",
  "sn": "Doe",
  "displayName": "John Doe",
  ...
  "ldapPassword": {
    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "cipher": "AES/CBC/PKCS5Padding",
        "stableId": "openidm-sym-default",
        "salt": "Ivmal5...",
        "data": "0mkywe...",
        "keySize": 16,
        "purpose": "idm.password.encryption",
        "iv": "ehSMbdNn...",
        "mac": "PssPOsW..."
      }
    }
  },
  ...
}
```

The password change succeeds.

Link Multiple Accounts to a Single Identity

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

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

Sample Overview

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

This sample uses two link qualifiers:

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

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

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

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

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

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

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

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

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

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

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

Prepare the Sample

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

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

Run the Sample

Create the Users, Roles, and Assignments

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

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

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

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

Note

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

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

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

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

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

Note

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

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

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

   1. The following command grants the Agent role to jdoe:

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/roles/-",
    "value": {
      "_ref": "managed/role/1b58ec8d-fae2-4b28-a5cf-b63567e4cf3f"
    }
  }
]' \
"http://localhost:8080/openidm/managed/user/02632173-e413-4af1-8495-f749d5880226"
{
  "_id": "02632173-e413-4af1-8495-f749d5880226",
  "_rev": "00000000dc6160c8",
  "displayName": "John Doe",
  "description": "Created for OpenIDM",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "1-415-599-1100",
  "sn": "Doe",
  "userName": "jdoe",
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": [
    {
    "_refResourceCollection": "managed/role",
    "_refResourceId": "1b58ec8d-fae2-4b28-a5cf-b63567e4cf3f",
    "_ref": "managed/role/1b58ec8d-fae2-4b28-a5cf-b63567e4cf3f"
    }
  ]
}
```

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

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/roles/-",
    "value": {
      "_ref": "managed/role/617368f2-fa4e-44a2-a25a-f0a86e16ef00"
    }
  }
]' \
"http://localhost:8080/openidm/managed/user/580f1441-ff8e-434b-9605-90e10a6fbdf6"
{
  "_id": "580f1441-ff8e-434b-9605-90e10a6fbdf6",
  "_rev": "000000004cab60c8",
  "displayName": "Barbara Jensen",
  "description": "Created for OpenIDM",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "telephoneNumber": "1-360-229-7105",
  "sn": "Jensen",
  "userName": "bjensen",
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": [
    {
      "_refResourceCollection": "managed/role",
      "_refResourceId": "617368f2-fa4e-44a2-a25a-f0a86e16ef00",
      "_ref": "managed/role/617368f2-fa4e-44a2-a25a-f0a86e16ef00"
    }
  ]
}
```

3. The following command grants the Insured role to jdoe:

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

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

4. Create the managed assignments.

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

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

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

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

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

5. Add the assignments to their respective roles.

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

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

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

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

Reconcile Managed Users to the LDAP Server

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

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

This reconciliation creates three new accounts in DS:

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

> **Note**
>
> Both users already exist in DS, from the `Example.ldif` file that you imported during the setup.

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

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

Link historical accounts

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

Sample overview

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

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

- `onLink-managedUser_systemLdapAccounts.js`

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

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

- `onUnlink-managedUser_systemLdapAccounts.js`

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

- `check_account_state_change.js`

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

- `ldapBackCorrelationQuery.js`

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

Run the sample

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

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

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

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

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

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

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

Because implicit synchronization is enabled by default, any change to the managed/user repository should be propagated to DS. For more information about implicit synchronization, refer to Synchronization types.

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

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

Note

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

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

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

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

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

- The date on which the accounts were linked
- The fact that this link is currently active
- The state of the account in DS (`enabled`)

6. Enable the liveSync schedule to propagate changes made in DS to the managed user repository.

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

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

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

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

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

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

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

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

   To do this by using the admin UI, select **Manage > User**, select **Joe Smith**'s account, and change his **Status** to *inactive* on his **Details** tab.

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

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

10. Request Joe Smith's historical accounts:

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

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

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

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

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

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

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

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

13. Request Joe Smith's historical accounts:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/24356bf0-f026-4dc1-9f68-2a571b0a236f/historicalAccounts?
_queryFilter=true"
{
  "result": [
    {
      "_id": "3f193422-156b-4b66-adcf-447db1b7d770",
      "_rev": "0000000037beefe7",
      "_ref": "system/ldap/account/da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa",
      "_refResourceCollection": "system/ldap/account",
      "_refResourceId": "da7c8fe9-4959-4dc9-9cd5-60c0ead9b0aa",
      "_refProperties": {
        "active": false,
        "stateLastChanged": "Mon May 18 2020 13:51:06 GMT+0200 (SAST)",
        "state": "disabled",
        "linkDate": "Mon May 18 2020 13:47:18 GMT+0200 (SAST)",
        "unlinkDate": "Mon May 18 2020 13:52:33 GMT+0200 (SAST)",
        "_id": "3f193422-156b-4b66-adcf-447db1b7d770",
        "_rev": "0000000037beefe7"
      }
    },
    {
      "_id": "8850640c-2233-4ddc-9725-6b4b2d59605f",
      "_rev": "000000000843ce68",
      "_ref": "system/ldap/account/52821eec-e00d-4321-8857-f46a870afc45",
      "_refResourceCollection": "system/ldap/account",
      "_refResourceId": "52821eec-e00d-4321-8857-f46a870afc45",
      "_refProperties": {
        "active": true,
        "stateLastChanged": "Mon May 18 2020 13:54:52 GMT+0200 (SAST)",
        "state": "enabled",
        "linkDate": "Mon May 18 2020 13:54:52 GMT+0200 (SAST)",
        "_id": "8850640c-2233-4ddc-9725-6b4b2d59605f",
        "_rev": "000000000843ce68"
      }
    }
  ],
  ...
}
```

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

Provision users with roles

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

Sample overview

IDM supports two types of roles:

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

Both provisioning roles and authorization roles use relationships to link the role object, and the managed object to which the role applies. For information about managing roles, refer to Managed Roles.

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

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

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

Prepare the sample

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

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

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

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

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

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

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

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

Note

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

3. Reconcile the repository.

The `sync.json` configuration file for this sample includes two mappings:

- `systemLdapAccounts_managedUser`, which synchronizes users from the source LDAP server with the target IDM repository.
- `managedUser_systemLdapAccounts`, which synchronizes changes from the repository with the LDAP server.

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

- To use the admin UI, select **Configure > Mapping**, click on the first mapping (**System/Ldap/Account → Managed User**), and click **Reconcile**.

- To use the REST interface, run the following command:

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

The sample is now ready to demonstrate provisioning roles.

Run the sample

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

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

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

Add assignments to a role definition

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

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

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

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

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

Tip

Display the roles in the admin UI by selecting **Manage > Role**.

2. Create a new managed assignment named Employee.

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-type: application/json" \
--request POST \
--data '{
   "name" : "Employee",
   "description": "Assignment for employees.",
   "mapping" : "managedUser_systemLdapAccounts",
   "attributes": [
     {
       "name": "employeeType",
       "value": [
         "Employee"
       ],
       "assignmentOperation" : "mergeWithTarget",
       "unassignmentOperation" : "removeFromTarget"
     }
   ]
 }' \
"http://localhost:8080/openidm/managed/assignment?_action=create"
{
  "_id": "1bbda95c-2a89-4e09-9719-8957849febeb",
  "_rev": "00000000ca15975d",
  "name": "Employee",
  "description": "Assignment for employees.",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "employeeType",
      "value": [
        "Employee"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ]
}
```

> **Tip**
>
> Create the assignment using the admin UI:
>
> 1. From the navigation bar, click **Manage > Assignment**.
> 2. On the **Assignment List** page, click **New Assignment**.
> 3. On the **New Assignment** page, do the following:
>   1. Enter the **Assignment Name**.
>   2. Enter the **Description**.
>   3. From the **Mapping** drop-down list, select the mapping for which the assignment is applied (`managedUser_systemLdapAccounts`).
>   4. Click **Save**.
> 4. Select the **Attributes** tab, and click **Add an Attribute**.
> 5. From the drop-down list, select **employeeType**.
> 6. In the adjacent *attribute value* area, click **item**.
> 7. From the **item 1** drop-down list, select **string**, and enter the value `Employee` .
> 8. Click **Save**.

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

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

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

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

> **Tip**
>
> Add the assignment to the role in the admin UI:
>
> 1. Select **Manage > Role**, and select the **Employee** role.
> 2. On the **Managed Assignments** tab, click **Add Managed Assignments**.
> 3. Select the **Employee** assignment and click **Add**.

Grant a role to a user and observe that user's role assignments

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

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

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

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

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

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

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

```
      "_refResourceId": "d4f6b571-7e71-4901-8033-090a15098867",
      "_ref": "managed/role/d4f6b571-7e71-4901-8033-090a15098867"
    }
  ]
}
```

Tip

Assign the role to bjensen by using the admin UI:

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

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=/
userName+eq+'bjensen'&_fields=_id,userName,roles,effectiveRoles,effectiveAssignments"
{
  "result": [
    {
      "_id": "57356103-a1d5-4aaa-bcc5-a640147704e0",
      "_rev": "000000005498d7b5",
      "userName": "bjensen",
      "effectiveAssignments": [
        {
          "_rev": "00000000ca15975d",
          "_id": "1bbda95c-2a89-4e09-9719-8957849febeb",
          "name": "Employee",
          "description": "Assignment for employees.",
          "mapping": "managedUser_systemLdapAccounts",
          "attributes": [
            {
              "name": "employeeType",
              "value": [
                "Employee"
              ],
              "assignmentOperation": "mergeWithTarget",
              "unassignmentOperation": "removeFromTarget"
            }
          ],
          "_refResourceCollection": "managed/assignment",
          "_refResourceId": "1bbda95c-2a89-4e09-9719-8957849febeb",
          "_ref": "managed/assignment1bbda95c-2a89-4e09-9719-8957849febeb"
        }
      ],
      "effectiveRoles": [
        {
          "_refResourceCollection": "managed/role",
          "_refResourceId": "d4f6b571-7e71-4901-8033-090a15098867",
          "_ref": "managed/role/d4f6b571-7e71-4901-8033-090a15098867"
        }
      ],
      "roles": [
        {
          "_ref": "managed/role/d4f6b571-7e71-4901-8033-090a15098867",
          "_refResourceCollection": "managed/role",
          "_refResourceId": "d4f6b571-7e71-4901-8033-090a15098867",
          "_refProperties": {
            "_id": "75441276-4ede-4655-855c-e13ed4b47e8e",
            "_rev": "00000000ccc59e6c"
          }
        }
```

```
      ]
    }
  ],
  ...
}
```

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

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

Propagate assignments to an external system

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

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

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

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

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

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

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

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

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

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

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

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

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

> **Tip**
>
> To add more attributes to the Employee assignment in the admin UI:
>
> 1. Select **Manage > Assignment**, and click the **Employee** assignment.
> 2. On the **Attributes** tab, select **Add an attribute**, and select the **ldapGroups** attribute.
> 3. Enter the following values, and click **Save**:
>
> ```
> cn=Employees,ou=Groups,dc=example,dc=com
> cn=Chat Users,ou=Groups,dc=example,dc=com
> ```

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

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

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

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account?_queryFilter=/
uid+sw+'bjensen'&_fields=dn,uid,employeeType,ldapGroups"
{
  "result": [
    {
      "_id": "887732e8-3db2-31bb-b329-20cd6fcecc05",
      "dn": "uid=bjensen,ou=People,dc=example,dc=com",
      "uid": "bjensen",
      "employeeType": [
        "Employee"
      ],
      "ldapGroups": [
        "cn=Chat Users,ou=Groups,dc=example,dc=com",
        "cn=openidm2,ou=Groups,dc=example,dc=com",
        "cn=Employees,ou=Groups,dc=example,dc=com"
      ]
    }
  ],
  ...
}
```

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

> **Tip**
>
> Use the admin UI to view bjensen's LDAP groups as follows:
>
> 1. Select **Manage > User**, and select **bjensen**.
> 2. On the **Linked Systems** tab, scroll down to the **ldapGroups** item.

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

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

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "name": "Contractor",
  "description": "Contractor assignment for contract workers.",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "ldapGroups",
      "value": [
        "cn=Contractors,ou=Groups,dc=example,dc=com"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    },
    {
      "name": "employeeType",
      "value": [
        "Contractor"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ]
}' \
"http://localhost:8080/openidm/managed/assignment?_action=create"
{
  "_id": "30323ed3-d885-4d09-94ca-c8f3b3408296",
  "_rev": "00000000db43da70",
  "name": "Contractor",
  "description": "Contractor assignment for contract workers.",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "ldapGroups",
      "value": [
        "cn=Contractors,ou=Groups,dc=example,dc=com"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    },
    {
      "name": "employeeType",
      "value": [
        "Contractor"
      ],
```

```
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ]
}
```

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

> **Tip**
>
> To create the assignment using the admin UI, refer to Add Assignments to a Role Definition.

5. Add the Contractor assignment to the Contractor role:

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

> **Tip**
>
> Add the Contractor assignment to the Contractor role in the admin UI, as follows:
>
> 1. Select **Manage > Role**, and select the **Contractor** role.
> 2. On the **Managed Assignments** tab, click **Add Managed Assignment**.
> 3. Select the **Contractor** assignment from **the dropdown list**, and click **Add**.

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

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

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

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

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

```
      ]
    }
  ],
  "effectiveRoles": [
    {
      "_refResourceCollection": "managed/role",
      "_refResourceId": "95899c38-e483-4d89-8aec-a88baab0603a",
      "_ref": "managed/role/95899c38-e483-4d89-8aec-a88baab0603a"
    }
  ]
}
```

Tip

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

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

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

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

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

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

Tip

Use the admin UI to view jdoe's LDAP groups as follows:

1. Select **Manage > User**, and select **jdoe**.
2. On the **Linked Systems tab**, scroll down to the **ldapGroups** item.

Note

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

For example, if the Employees group was a dynamic group, membership might be set based on the `employeeType` attribute directly, by setting the `memberURL` in the group to `ldap:///ou=People,dc=example,dc=com??sub?(employeeType=Employee)`. You would then remove the `ldapGroups` attribute from the Employee assignment, since group membership is handled by `employeeType`.

This membership won't be listed in the `ldapGroups` attribute in IDM (since it is no longer set there), but can be verified by querying DS directly:

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

For more information about dynamic groups in DS, refer to Dynamic Groups in the *Configuration Guide* for ForgeRock Directory Services.

Remove a role grant from a user and observe that user's role assignments

In this section, you will remove the Contractor role from jdoe's managed user entry and observe the subsequent change to jdoe's managed assignments, and to the corresponding attributes in DS.

1. Before you change jdoe's roles, view his entry again to examine his current roles:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=/userName+eq+'jdoe'&_fields=_id,roles"
{
  "result": [
    {
      "_id": "0d8a1d10-62e0-43aa-9892-ec51258771d0",
      "_rev": "00000000aa64591e",
      "roles": [
        {
          "_ref": "managed/role/95899c38-e483-4d89-8aec-a88baab0603a",
          "_refResourceCollection": "managed/role",
          "_refResourceId": "95899c38-e483-4d89-8aec-a88baab0603a",
          "_refProperties": {
            "_id": "de68f5d9-baf9-49ca-8db5-96f0a382946d",
            "_rev": "00000000e299a021"
          }
        }
      ]
    }
  ],
  ...
}
```

Note the following IDs in this example output, you need them in the next step:

- The ID of jdoe's user object is `0d8a1d10-62e0-43aa-9892-ec51258771d0`.
- The ID of the contractor role (`_refResourceId`) is `95899c38-e483-4d89-8aec-a88baab0603a`.
- The ID of the *relationship* that expresses the role grant is `de68f5d9-baf9-49ca-8db5-96f0a382946d`.

> **Tip**
>
> View jdoe's current roles in the admin UI:
>
> 1. Select **Manage > User**, and select **jdoe**.
> 2. The **Provisioning Roles** tab lists the current roles.

2. Remove the Contractor role from jdoe's entry by sending a DELETE request to his user entry, specifying the *relationship ID*:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/managed/user/0d8a1d10-62e0-43aa-9892-ec51258771d0/roles/de68f5d9-baf9-49ca-8db5-96f0a382946d"
{
  "_id": "de68f5d9-baf9-49ca-8db5-96f0a382946d",
  "_rev": "00000000e299a021",
  "_ref": "managed/role/95899c38-e483-4d89-8aec-a88baab0603a",
  "_refResourceCollection": "managed/role",
  "_refResourceId": "95899c38-e483-4d89-8aec-a88baab0603a",
  "_refProperties": {
    "_id": "de68f5d9-baf9-49ca-8db5-96f0a382946d",
    "_rev": "00000000e299a021"
  }
}
```

The output shows that the *relationship* between the user and the role was deleted.

> **Tip**
>
> Use the admin UI to remove the Contractor role from jdoe's entry as follows:
>
> 1. Select **Manage > User**, and select **jdoe**.
> 2. On the **Provisioning Roles** tab, check the **box** next to the Contractor role and click **Remove Selected Provisioning Roles**.

3. Verify jdoe's `employeeType` and `ldapGroups`.

   The removal of the Contractor role causes a synchronization operation to be run on jdoe's entry. His `employeeType` and `ldapGroups` attributes in DS should be reset to what they were before he was granted the Contractor role.

   Check jdoe's attributes by querying his object in the LDAP directory, over the REST interface:

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

Tip

Use the admin UI to view jdoe's LDAP groups as follows:

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

For more information about roles, assignments, and how to manipulate them, refer to Managed Roles.

Provision users with workflow

This sample demonstrates a typical workflow use case, provisioning new users.

The sample uses the admin UI to set up the initial users and roles, then shows how users can complete their registration process in the End User UI.

The sample simulates the following scenario:

- An existing employee requests that an outside contractor be granted access to an organization's system.
- The *system* in this case, is the IDM managed user repository and a remote HR data source, represented by a CSV file (`hr.csv`).
- User roles are stored separately, in a second CSV file (`roles.csv`).

The sample has three mappings—two for the bidirectional synchronization of the managed user repository and the HR data store, and one for the synchronization of the roles data to the managed repository.

Prepare the sample

In this section, you start IDM, configure the outbound email service, and reconcile user and role data. The reconciliation operations create two managed users, `user1` and `manager1`, and two managed roles, `employee` (assigned to `user1`) and `manager` (assigned to `manager1`).

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

1. Edit the `/path/to/openidm/samples/provisioning-with-workflow/conf/datasource.jdbc-default.json` file with the details of your JDBC repository. For more information, refer to Select a repository.
2. Start IDM with the configuration for the provisioning sample:

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

3. Log in to the admin UI.
4. Configure the outbound email service:
    1. From the navigation bar, click **Configure > Email Settings**.
    2. On the **Email Settings** page, enable the outbound mail service, enter the connection information, and click **Save**.
5. Enable Password Reset:
    1. From the navigation bar, click **Configure > Password Reset**.
    2. On the **Password Reset** page, enable password reset, enter the applicable information, and click **Save**.

    > **Note**
    >
    > For additional password reset information, refer to Email for password reset.

6. Reconcile the role and user data:
    1. From the navigation bar, click **Configure > Mappings**.
    2. Select the first mapping (**systemRolesFileRole_internalRole**), and click **Reconcile**.
    3. To verify the reconciliation:
        1. From the navigation bar, click **Manage > Role**.
        2. On the **Roles** page, click the **Internal** tab.

        IDM displays the two roles created in the previous step:

        - `employee`
        - `manager`
    4. From the navigation bar, click **Configure > Mappings**.

5. Select the second mapping (**systemCsvfileAccounts_managedUser**), and click **Reconcile**.

   The reconciliation operation creates the top-level managers (users who do not have their own `manager` property) in the managed user repository. In this sample, there is only one top-level manager (`manager1`).

6. Select the second mapping again (**systemCsvfileAccounts_managedUser**), and click **Reconcile**.

   This reconciliation operation creates the employees of the managers that were created by the previous reconciliation. In this sample, there is only one employee (`employee1`).

7. From the navigation bar, click **Manage > User**, and verify the users `manager1` and `user1` exist.

7. Verify the relationships between the new user and role objects:

   1. Click **user1**.

      The **Manager** field displays `manager1` for this user.

   2. Click the **Authorization Roles** tab.

      `user1` has two roles, `openidm-authorized` and `employee`.

   3. From the breadcrumb link at the top of the page, click **User**, and select `manager1`.

      The **Manager** field is empty for this user.

   4. Click the **Authorization Roles** tab.

      `manager1` has three roles: `manager`, `openidm-authorized`, and `openidm-tasks-manager`.

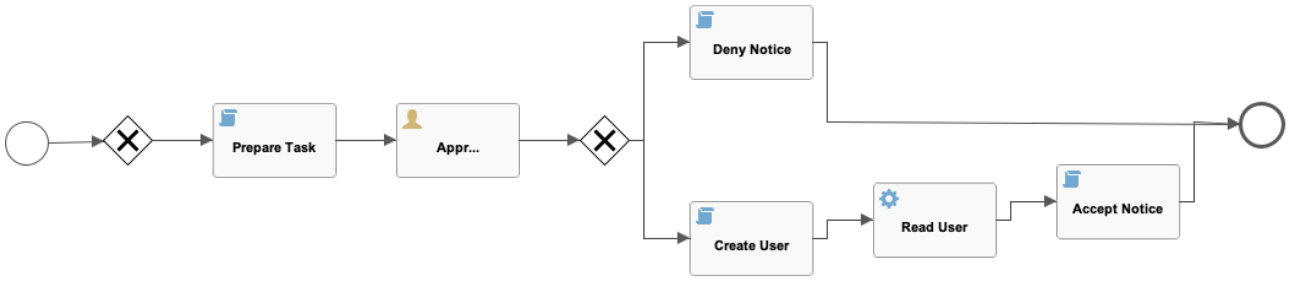
8. Verify the available workflows:

   1. From the navigation bar, click **Manage > Processes**.

   2. On the **Workflow Processes** page, select the **Definitions** tab.

   3. From the **Definitions** list, click **Contractor onboarding process**.

      IDM displays a diagram similar to the following:



9. Log out of the admin UI.

Run the sample

During this part of the sample, an existing employee initiates a *Contractor Onboarding* process. This process is a request to add a contractor to the managed user repository, with an option to include the contractor in the original HR data source (`hr.csv`).

When the employee has completed the required form, the request is sent to the manager for approval. Any user with the role `manager` can claim the approval task. If the request is approved, an email is sent to the address provided in the initial form, with a request for the contractor to reset their password. When the password reset has been completed, the contractor is created in the managed user repository. If a request was made to add the contractor to the original HR data source, this is done when the manager approves the request.

1. Log in to the End User UI (`https://localhost:8443/`) as the user you created in the previous section (`user1`) with password `Welcome1` .
2. Navigate to the dashboard, with the **Dashboard icon** (). Alternatively, select the **Menu icon** (), and select **Dashboard**.
3. Initiate the provisioning workflow as `user1` :
    1. Scroll down to the **Start a Process** menu, and click **Edit** adjacent to **Contractor onboarding process**.
    2. Complete the form for the sample user you will be creating. Use an accessible email address, as you'll need the email message to complete this workflow.
    3. Enable **Create in CSV File**. This option enables implicit synchronization from the managed user repository to the `hr.csv` file.

        > **Note**
        >
        > `user1` does not provide a password for this user. A password reset request is sent to the email address provided on this form to ensure that only the actual contractor can log in with this account.

    4. Select **Submit** to initiate the process.
    5. Log out of the End User UI.
4. Approve the workflow task as `manager1` :
    1. Log in to the End User UI as `manager1` with password `Welcome1` .
    2. Navigate to the **dashboard**, with the **Dashboard icon** (). Alternatively, select the **Menu icon** (), and select **Dashboard**.
    3. Under **Unassigned Tasks**, locate the **Approve Contractor** task, select **Assign**, and click **Assign to Me**.

        Approve Contractor is now listed under **My Tasks**.
    4. Click **Edit** adjacent to the task name.
    5. Review the form content, and click **Accept**.

        > **Note**
        >
        > This is the same content you provided as `user1`.

    6. Log out of the End User UI.
5. Verify that the contractor has been created in the HR data source (`/path/to/openidm/samples/provisioning-with-workflow/data/hr.csv`):

```
"username","firstname","lastname","manager", "department","jobTitle",     ...
"user1",   "Ordinary", "Employee","manager1","dep1",      "job1",         ...
"manager1","Big",      "Manager", "",        "dep1",      "Manager",      ...
"bjensen", "Barbara",  "Jensen",  "user1",   "Payroll",   "Payroll clerk",...
```

Note the addition of the new contractor entry, `bjensen`.

6. Complete the password reset process:

    1. Check the email account you provided on the initial form for a message with the subject line "Reset your password".
    2. Open the password reset email, and click **Password reset link**.

       The link takes you to the End User UI.
    3. Click **Reset Your Password**.
    4. Enter a new password, and click **Change Password**.

       The password that you enter here must comply with the password policy that is configured for managed users. For more information, refer to Enforcing Password Policy.
    5. Click **Sign In**, and enter the username and new password.
    6. Click the **notifications icon** (🔔), and you should see a *welcome* message.

> **Important**
>
> If you declined the approval request, the user is not created in either the managed user repository, or in the HR CSV file.

Connect to DS with ScriptedREST

This sample uses the scripted REST connector to interact with the ForgeRock Directory Services (DS) REST API, using Groovy scripts. The sample demonstrates reconciliation, implicit sync, and liveSync between the IDM repository and a DS instance.

The scripted REST connector is bundled with IDM in the JAR `openidm/connectors/scriptedrest-connector-1.5.20.14.jar`.

The Groovy scripts required for the sample are located in the `samples/scripted-rest-with-dj/tools` directory. You must customize these scripts to address the requirements of your specific deployment; however, the sample scripts are a good starting point on which to base your customization.

> **Important**
>
> The Rest2ldap HTTP endpoint provided with DS is an evolving interface. As such, compatibility between versions is not guaranteed. This sample was tested with DS 7.3.

Set up DS

1. Set up DS, but remove the line `--set ds-user-data/ldifFile:Example.ldif`.

2. Optionally, you can enable an HTTP access logger on the DS server.

3. Import the data required for the sample:

```
/path/to/opendj/bin/ldapmodify \
--port 1636 \
--useSSL \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--hostname localhost \
--bindDN uid=admin \
--bindPassword password \
--filename /path/to/openidm/samples/scripted-rest-with-dj/data/ldap.ldif
# ADD operation successful for DN dc=example,dc=com

# ADD operation successful for DN ou=Administrators,dc=example,dc=com

# ADD operation successful for DN uid=idm,ou=Administrators,dc=example,dc=com

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

# ADD operation successful for DN ou=Groups,dc=example,dc=com
```

4. Set up the access control instructions (ACIs) that enable the IDM administrator user to read the DS external change log:

```
/path/to/opendj/bin/dsconfig set-access-control-handler-prop \
--port 4444 \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--hostname localhost \
--bindDN uid=admin \
--bindPassword password \
--add global-aci:"(target=\"ldap:///cn=changelog\")(targetattr=\"*||+\") \
(version 3.0; acl \"IDM can access cn=changelog\"; \
allow (read,search,compare) \
userdn=\"ldap:///uid=idm,ou=Administrators,dc=example,dc=com\";)" \
--no-prompt
```

```
/path/to/opendj/bin/dsconfig set-access-control-handler-prop \
--port 4444 \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--hostname localhost \
--bindDN uid=admin \
--bindPassword password \
--add global-aci:"(targetcontrol=\"1.3.6.1.4.1.26027.1.5.4\") \
(version 3.0; acl \"IDM changelog control access\"; \
allow (read) \
userdn=\"ldap:///uid=idm,ou=Administrators,dc=example,dc=com\";)" \
--no-prompt
```

5. Enable the default Rest2ldap HTTP endpoint:

```
/path/to/opendj/bin/dsconfig set-http-endpoint-prop \
--port 4444 \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--hostname localhost \
--bindDN uid=admin \
--bindPassword password \
--endpoint-name /api \
--set authorization-mechanism:"HTTP Basic" \
--set config-directory:config/rest2ldap/endpoints/api \
--set enabled:true \
--no-prompt
```

For more information, refer to HTTP User APIs in the *DS Configuration Guide*.

6. Replace the default DS REST to LDAP configuration with the configuration for this sample:

```
cp /path/to/openidm/samples/scripted-rest-with-dj/data/example-v1.json /path/to/opendj/config/
rest2ldap/endpoints/api/
```

7. Restart DS for the configuration change to take effect.

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

Run the sample

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

1. Start IDM with the configuration for the ScriptedREST sample:

```
cd /path/to/openidm/
./startup.sh -p samples/scripted-rest-with-dj
```

2. Check that the scripted REST connector can reach the DS instance by obtaining the connector status over REST:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/scriptedrest?_action=test"
{
  "name": "scriptedrest",
  "enabled": true,
  "config": "config/provisioner.openicf/scriptedrest",
  "connectorRef": {
    "bundleVersion": "[1.5.0.0,1.6.0.0)",
    "bundleName": "org.forgerock.openicf.connectors.scriptedrest-connector",
    "connectorName": "org.forgerock.openicf.connectors.scriptedrest.ScriptedRESTConnector"
  },
  "displayName": "Scripted REST Connector",
  "objectTypes": [
    "__ALL__",
    "account",
    "group"
  ],
  "ok": true
}
```

3. Create a group entry on the DS server:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "cn": "group1"
}' \
"http://localhost:8080/openidm/system/scriptedrest/group?_action=create"
{
  "_id": "group1",
  "members": null,
  "created": "2020-07-21T23:04:25Z",
  "cn": "group1",
  "displayName": "group1"
}
```

4. Create a user entry on the DS server. This command creates a user with `uid` scarter:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "givenName": "Steven",
  "familyName": "Carter",
  "emailAddress": "scarter@example.com",
  "telephoneNumber": "444-444-4444",
  "password": "5up35tr0ng",
  "displayName": "Steven.Carter",
  "uid": "scarter"
}' \
"http://localhost:8080/openidm/system/scriptedrest/account?_action=create"
{
  "_id": "scarter",
  "givenName": "Steven",
  "groups": null,
  "displayName": "Steven.Carter",
  "emailAddress": "scarter@example.com",
  "uid": "scarter",
  "created": "2020-07-21T23:07:13Z",
  "familyName": "Carter",
  "telephoneNumber": "444-444-4444"
}
```

Notice that the user is not a member of any group.

5. Reconcile the DS server with the managed user repository:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemRestLdapUser_managedUser&waitForCompletion=true"
{
  "_id": "ee7534bd-ccfd-4f6a-bdc3-49caa6d2043c-547",
  "state": "SUCCESS"
}
```

6. The reconciliation creates a managed user with a server-assigned ID. To retrieve the ID, run the following query:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true&_fields=_id"
{
  "result": [
    {
      "_id": "4657a420-6608-410e-baa7-f64668cc500c",
      "_rev": "000000007995f006"
    }
  ],
  ...
}
```

7. To initialize liveSync set the sync token by running one liveSync operation over REST:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/scriptedrest/account?_action=liveSync"
{
  "connectorData": {
    "nativeType": "string",
    "syncToken": "8"
  }
}
```

8. Update Steven Carter's managed user entry, by modifying his telephone number. Specify the user ID that you retrieved previously:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "replace",
    "field": "telephoneNumber",
    "value": "555-555-5555"
  }
]' \
"http://localhost:8080/openidm/managed/user/4657a420-6608-410e-baa7-f64668cc500c"
{
  "_id": "4657a420-6608-410e-baa7-f64668cc500c",
  "_rev": "0000000096edf021",
  "userName": "scarter",
  "mail": "scarter@example.com",
  "displayName": "Steven.Carter",
  "telephoneNumber": "555-555-5555",
  "givenName": "Steven",
  "sn": "Carter",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

9. The implicit synchronization mechanism between the managed user repository and DS propagates the change to DS. You can check this change by reading scarter's user entry in DS and noting the changed `telephoneNumber`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/scriptedrest/account/scarter"
{
  "_id": "scarter",
  "familyName": "Carter",
  "givenName": "Steven",
  "created": "2018-02-07T10:14:31Z",
  "uid": "scarter",
  "groups": null,
  "emailAddress": "scarter@example.com",
  "displayName": "Steven.Carter",
  "telephoneNumber": "555-555-5555"
}
```

10. Now, update Steven Carter's entry on the DS server, by modifying the `givenName`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "replace",
    "field": "givenName",
    "value": "Steve"
  }
]' \
"http://localhost:8080/openidm/system/scriptedrest/account/scarter"
{
  "_id": "scarter",
  "givenName": "Steve",
  "groups": null,
  "displayName": "Steven.Carter",
  "emailAddress": "scarter@example.com",
  "uid": "scarter",
  "created": "2020-07-21T23:07:13Z",
  "familyName": "Carter",
  "telephoneNumber": "555-555-5555"
}
```

11. To propagate the change made on DS back to the managed user entry, launch a liveSync operation, either by defining a schedule, or directly over REST. The following command launches liveSync over REST:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/scriptedrest/account?_action=liveSync"
{
  "connectorData": {
    "nativeType": "string",
    "syncToken": "9"
  },
  "_rev": "000000000a585336",
  "_id": "SYSTEMSCRIPTEDRESTACCOUNT"
}
```

12. Verify that the changes propagated by reading scarter's managed user entry and noting the changed `givenName`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/4657a420-6608-410e-baa7-f64668cc500c"
{
  "_id": "4657a420-6608-410e-baa7-f64668cc500c",
  "_rev": "000000007937efb7",
  "userName": "scarter",
  "mail": "scarter@example.com",
  "displayName": "Steven.Carter",
  "telephoneNumber": "555-555-5555",
  "givenName": "Steve",
  "sn": "Carter",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

13. Add user scarter to the group you created previously, by updating the group entry:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--header "If-Match: *" \
--request PUT \
--data '{
  "_id": "group1",
  "members": [{"_id": "scarter"}]
}' \
"http://localhost:8080/openidm/system/scriptedrest/group/group1"
{
  "_id": "group1",
  "displayName": "group1",
  "created": "2018-02-07T10:14:12Z",
  "members": [
    {
      "_id": "scarter",
      "displayName": "Steven.Carter"
    }
  ],
  "cn": "group1",
  "lastModified": "2018-02-07T10:20:22Z"
}
```

14. Read Steven Carter's user entry in DS, to verify that he is now a member of group1:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/scriptedrest/account/scarter"
{
  "_id": "scarter",
  "givenName": "Steve",
  "groups": [
    {
      "_id": "group1"
    }
  ],
  "displayName": "Steven.Carter",
  "emailAddress": "scarter@example.com",
  "uid": "scarter",
  "created": "2020-07-21T23:07:13Z",
  "familyName": "Carter",
  "telephoneNumber": "555-555-5555"
}
```

15. Read the group entry to verify its members:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/scriptedrest/group/group1"
{
  "_id": "group1",
  "lastModified": "2020-07-21T23:17:09Z",
  "members": [
    {
      "_id": "scarter",
      "displayName": "Steven.Carter"
    }
  ],
  "created": "2020-07-21T23:04:25Z",
  "cn": "group1",
  "displayName": "group1"
}
```

16. Reconcile the DS groups with the managed group repository:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemRestLdapGroup_managedGroup&waitForCompletion=true"
{
  "_id": "ee7534bd-ccfd-4f6a-bdc3-49caa6d2043c-1477",
  "state": "SUCCESS"
}
```

17. The reconciliation creates a managed group with a server-assigned ID. To retrieve the group ID, run the following query:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request GET \
"http://localhost:8080/openidm/managed/group?_queryId=query-all-ids"
{
  "result": [
    {
      "_id": "67b5ec50-d5a6-4bfa-bb19-17965447ad00",
      "_rev": "00000000b0e95e9b"
    }
  ],
  ...
}
```

18. Read the managed group to verify that the DS group has been added, and that its members have been reconciled to the managed group repository. Specify the ID that you retrieved in the previous step:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/group/67b5ec50-d5a6-4bfa-bb19-17965447ad00"
{
  "_id": "67b5ec50-d5a6-4bfa-bb19-17965447ad00",
  "_rev": "00000000b0e95e9b",
  "members": [
    {
      "_id": "scarter",
      "displayName": "Steven.Carter"
    }
  ],
  "displayName": "group1"
}
```

19. Delete the DS user and group entries, returning the DS server to its initial state.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/system/scriptedrest/account/scarter"
{
  "_id": "scarter",
  "givenName": "Steve",
  "groups": [
    {
      "_id": "group1"
    }
  ],
  "displayName": "Steven.Carter",
  "emailAddress": "scarter@example.com",
  "uid": "scarter",
  "created": "2020-07-21T23:07:13Z",
  "familyName": "Carter",
  "telephoneNumber": "555-555-5555"
}
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/system/scriptedrest/group/group1"
{
  "_id": "group1",
  "lastModified": "2020-07-21T23:17:09Z",
  "members": null,
  "created": "2020-07-21T23:04:25Z",
  "cn": "group1",
  "displayName": "group1"
}
```

Connect to Active Directory with the PowerShell connector

This sample shows an implementation of the PowerShell Connector toolkit and provides a number of PowerShell scripts that let you perform basic CRUD (create, read, update, delete) operations on an Active Directory server.

The sample uses the MS Active Directory PowerShell module. For more information on this module, refer to the corresponding Microsoft documentation.

Sample overview

The generic *PowerShell Connector Toolkit* enables you to run PowerShell scripts on any external resource. The PowerShell Connector Toolkit is not a complete connector, in the traditional sense. Rather, it is a framework within which you must write your own PowerShell scripts to address the requirements of your Microsoft Windows ecosystem. You can use the PowerShell Connector Toolkit to create connectors that can provision any Microsoft system.

The PowerShell Connector Toolkit is available from the Backstage download site.

This sample assumes that IDM is running on a Windows system on the localhost. It also assumes that Active Directory and the OpenICF .NET connector server run on a remote Windows server. The PowerShell connector runs on the .NET connector server.

To use this sample for IDM instances installed on UNIX systems, adjust the relevant commands shown with PowerShell prompts.

Note

By default, the `Get-ADUser` and `Get-ADGroup` cmdlets are not thread safe. To avoid thread issues when you use this connector with Active Directory, you must set the pooling configuration properties as follows:

```
"UseInterpretersPool" : true,
"MinInterpretersPoolSize" : 1,
"MaxInterpretersPoolSize" : 10
```

For more information about these properties, refer to Configure the PowerShell Connector.

Prepare the sample

Run the commands in this procedure from the PowerShell command line. The continuation character used in the command is the back-tick (`).

1. Install, configure, and start the .NET connector server on the machine that is running an Active Directory Domain Controller or on a workstation on which the Microsoft Active Directory PowerShell module is installed.

   For instructions on installing the .NET connector server, refer to Set Up a .NET RCS.

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

3. Download the PowerShell Connector Toolkit archive (`mspowershell-connector-1.4.7.0.zip`) from the Backstage download site.

4. Extract the archive and move the `MsPowerShell.Connector.dll` to the folder in which the connector server application (`connectorserver.exe`) is located.

5. Copy the PowerShell scripts and the ADSISearch module from the `samples\scripted-powershell-with-ad\tools` directory, to the machine on which the connector server is installed.

```
dir \path\to\openidm\samples\scripted-powershell-with-ad\tools
Directory: C:\path\to\openidm\samples\scripted-powershell-with-ad\tools

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         4/3/2018   3:26 AM           4279 ADAuthenticate.ps1
-a----         4/3/2018   3:26 AM           9055 ADCreate.ps1
-a----         4/3/2018   3:26 AM           3717 ADDelete.ps1
-a----         4/3/2018   3:26 AM          10756 ADSchema.ps1
-a----         4/3/2018   3:26 AM           4625 ADSearch.ps1
-a----         4/3/2018   3:26 AM           8064 ADSISearch.psm1
-a----         4/3/2018   3:26 AM           5918 ADSync.ps1
-a----         4/3/2018   3:26 AM           2408 ADTest.ps1
-a----         4/3/2018   3:26 AM          18406 ADUpdate.ps1
PS C:\path\to\openidm>
```

6. Copy the sample connector configuration for the PowerShell connector to your project's `conf` directory.

```
cp \path\to\openidm\samples\example-configurations\provisioners\provisioner.openicf-
adpowershell.json \path\to\openidm\conf
```

   The following excerpt of the sample connector configuration shows the configuration properties:

```
"configurationProperties" : {
    "AuthenticateScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADAuthenticate.ps1",
    "CreateScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADCreate.ps1",
    "DeleteScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADDelete.ps1",
    "SchemaScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADSchema.ps1",
    "SearchScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADSearch.ps1",
    "SyncScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADSync.ps1",
    "TestScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADTest.ps1",
    "UpdateScriptFileName" : "C:/openidm/samples/scripted-powershell-with-ad/tools/ADUpdate.ps1",
    "VariablesPrefix" : "Connector",
    "QueryFilterType" : "Ldap",
    "ReloadScriptOnExecution" : true,
    "UseInterpretersPool" : true,
    "SubstituteUidAndNameInQueryFilter" : true,
    "UidAttributeName" : "ObjectGUID",
    "NameAttributeName" : "DistinguishedName",
    "PsModulesToImport" : [
        "ActiveDirectory",
        "C:/openidm/samples/scripted-powershell-with-ad/tools/ADSISearch.psm1"
    ],
    "Host" : "",
    "Port" : null,
    "Login" : "",
    "Password" : null,
    "CustomProperties" : ["baseContext = CN=Users,DC=example,DC=com" ],
    "MinInterpretersPoolSize" : 1,
    "MaxInterpretersPoolSize" : 10
},
```

The sample connector configuration assumes that the scripts are located in `C:/openidm/samples/scripted-powershell-with-ad/tools/`. If you copied your scripts to a different location, or are using a different base context for search and synchronization operations such as `DC=example,DC=org`, adjust your connector configuration file accordingly.

Important

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

The host, port, login and password of the machine on which Active Directory runs do not need to be specified here. By default the Active Directory cmdlets pick up the first available Domain Controller. In addition, the scripts are executed using the credentials of the .Net connector server.

Note

The `ReloadScriptOnExecution` property is set to `true` in this sample configuration. This setting causes script files to be reloaded each time the script is invoked. Having script files reloaded each time is suitable for debugging purposes. However, this property should be set to `false` in production environments, as the script reloading can have a negative performance impact.

7. Make sure that the value of the `connectorHostRef` property in the connector configuration file matches the value that you specified in the remote connector configuration file, in step 2 of this procedure. For example:

```
 "connectorHostRef" : "dotnet",
```

Run the Sample

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

```
\path\to\openidm\startup.bat
```

When IDM has started, test the sample by using the `curl` command-line utility. The following examples test the scripts that were provided in the `tools` directory.

1. Test the connector configuration, and whether IDM is able to connect to the .NET connector server with the following request:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0"  `
--request POST `
"http://localhost:8080/openidm/system?_action=test"
[
  {
    "ok": true,
    "connectorRef": {
      "bundleVersion": "[1.4.2.0,1.5.0.0)",
      "bundleName": "MsPowerShell.Connector",
      "connectorName": "Org.ForgeRock.OpenICF.Connectors.MsPowerShell.MsPowerShellConnector"
    },
    "objectTypes": [
      "__ALL__",
      "group",
      "account"
    ],
    "config": "config/provisioner.openicf/adpowershell",
    "enabled": true,
    "name": "adpowershell"
  }
]
```

2. Query the users in your Active Directory with the following request:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0"  `
--request GET `
"http://localhost:8080/openidm/system/adpowershell/account?_queryId=query-all-ids"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 1257,
  "result": [
    {
      "_id": "7c41496a-9898-4074-a537-bed696b6be92"
    },
    {
      "_id": "f2e08a5c-473f-4798-a2d5-d5cc27c862a9"
    },
    {
      "_id": "6feef4a0-b121-43dc-be68-a96703a49aba"
    },
...
```

3. To return the complete record of a specific user, include the ID of the user in the URL. The following request returns the record for Steven Carter:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0"  `
--request GET `
"http://localhost:8080/openidm/system/adpowershell/account/6feef4a0-b121-43dc-be68-a96703a49aba"
{
  "_id": "6feef4a0-b121-43dc-be68-a96703a49aba",
  "postalCode": null,
  "passwordNotRequired": false,
  "cn": "Steven Carter",
  "name": "Steven Carter",
  "trustedForDelegation": false,
  "uSNChanged": "47219",
  "manager": null,
  "objectGUID": "6feef4a0-b121-43dc-be68-a96703a49aba",
  "modifyTimeStamp": "11/27/2014 3:37:16 PM",
  "employeeNumber": null,
  "sn": "Carter",
  "userAccountControl": 512,
  "passwordNeverExpires": false,
  "displayName": "Steven Carter",
  "initials": null,
  "pwdLastSet": "130615726366949784",
  "scriptPath": null,
  "badPasswordTime": "0",
  "employeeID": null,
  "badPwdCount": "0",
  "accountExpirationDate": null,
  "userPrincipalName": "steve.carter@ad0.example.com",
  "sAMAccountName": "steve.carter",
  "mail": "steven.carter@example.com",
  "logonCount": "0",
  "cannotChangePassword": false,
  "division": null,
  "streetAddress": null,
  "allowReversiblePasswordEncryption": false,
  "description": null,
  "whenChanged": "11/27/2014 3:37:16 PM",
  "title": null,
  "lastLogon": "0",
  "company": null,
  "homeDirectory": null,
  "whenCreated": "6/23/2014 2:50:48 PM",
  "givenName": "Steven",
  "telephoneNumber": "555-2518",
  "homeDrive": null,
  "uSNCreated": "20912",
  "smartcardLogonRequired": false,
  "distinguishedName": "CN=Steven Carter,CN=Users,DC=example,DC=com",
  "createTimeStamp": "6/23/2014 2:50:48 PM",
  "department": null,
```

```
  "memberOf": [
    "CN=employees,DC=example,DC=com"
  ],
  "homePhone": null
}
```

4. Test that you can authenticate as one of the users in your Active Directory. The username that you specify here can be either an ObjectGUID, UPN, sAMAccountname or CN:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0"  `
--header "Content-Type: application/json" `
--request POST `
--data "{
     \"username\" : \"Steven Carter\",
     \"password\" : \"Passw0rd\"
     }" `
"http://localhost:8080/openidm/system/adpowershell/account?_action=authenticate"
{
  "_id": "6feef4a0-b121-43dc-be68-a96703a49aba"
}
```

   The request returns the ObjectGUID if the authentication is successful.

5. You can return the complete record for a specific user, using the query filter syntax described in Construct Queries.

   The following query returns the record for the guest user:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0"  `
--request GET `
"http://localhost:8080/openidm/system/adpowershell/account?_queryFilter=cn+eq+'guest'"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 1,
  "result": [
    {
      "_id": "f2e08a5c-473f-4798-a2d5-d5cc27c862a9",
      "postalCode": null,
      "passwordNotRequired": true,
      "cn": "Guest",
      "name": "Guest",
      "trustedForDelegation": false,
      "uSNChanged": "8197",
      "manager": null,
      "objectGUID": "f2e08a5c-473f-4798-a2d5-d5cc27c862a9",
      "modifyTimeStamp": "6/9/2014 12:35:16 PM",
      "employeeNumber": null,
      "userAccountControl": 66082,
      "whenChanged": "6/9/2014 12:35:16 PM",
      "initials": null,
      "pwdLastSet": "0",
      "scriptPath": null,
      "badPasswordTime": "0",
      "employeeID": null,
      "badPwdCount": "0",
      "accountExpirationDate": null,
      "sAMAccountName": "Guest",
      "logonCount": "0",
      "cannotChangePassword": true,
      "division": null,
      "streetAddress": null,
      "allowReversiblePasswordEncryption": false,
      "description": "Built-in account for guest access to the computer/domain",
      "userPrincipalName": null,
      "title": null,
      "lastLogon": "0",
      "company": null,
      "homeDirectory": null,
      "whenCreated": "6/9/2014 12:35:16 PM",
      "givenName": null,
      "homeDrive": null,
      "uSNCreated": "8197",
      "smartcardLogonRequired": false,
      "distinguishedName": "CN=Guest,CN=Users,DC=example,DC=com",
      "createTimeStamp": "6/9/2014 12:35:16 PM",
      "department": null,
      "memberOf": [
        "CN=Guests,CN=Builtin,DC=example,DC=com"
```

```
      ],
      "homePhone": null,
      "displayName": null,
      "passwordNeverExpires": true
    }
  ]
}
```

6. Test that you can create a user on the Active Directory server by sending a POST request with the `create` action.

   The following request creates the user `Jane Doe` on the Active Directory server:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0"  `
--header "Content-Type: application/json" `
--request POST `
--data '{
  \"distinguishedName\" : \"CN=Jane Doe,CN=Users,DC=EXAMPLE,DC=COM\",
  \"sn\" : \"Doe\",
  \"cn\" : \"Jane Doe\",
  \"sAMAccountName\" : \"sample\",
  \"userPrincipalName\" : \"janedoe@example.com\",
  \"enabled\" : true,
  \"password\" : \"Passw0rd\",
  \"telephoneNumber\" : \"0052-611-091\"
 }' `
"http://localhost:8080/openidm/system/adpowershell/account?_action=create"
{
  "_id": "42725210-8dce-4fdf-b0e0-393cf0377fdf",
  "title": null,
  "uSNCreated": "47244",
  "pwdLastSet": "130615892934093041",
  "cannotChangePassword": false,
  "telephoneNumber": "0052-611-091",
  "smartcardLogonRequired": false,
  "badPwdCount": "0",
  "department": null,
  "distinguishedName": "CN=Jane Doe,CN=Users,DC=example,DC=com",
  "badPasswordTime": "0",
  "employeeID": null,
  "cn": "Jane Doe",
  "division": null,
  "description": null,
  "userPrincipalName": "janedoe@example.com",
  "passwordNeverExpires": false,
  "company": null,
  "memberOf": [],
  "givenName": null,
  "streetAddress": null,
  "sn": "Doe",
  "initials": null,
  "logonCount": "0",
  "homeDirectory": null,
  "employeeNumber": null,
  "objectGUID": "42725210-8dce-4fdf-b0e0-393cf0377fdf",
  "manager": null,
  "lastLogon": "0",
  "trustedForDelegation": false,
  "scriptPath": null,
  "allowReversiblePasswordEncryption": false,
  "modifyTimeStamp": "11/27/2014 8:14:53 PM",
  "whenCreated": "11/27/2014 8:14:52 PM",
  "whenChanged": "11/27/2014 8:14:53 PM",
  "accountExpirationDate": null,
```

```
  "name": "Jane Doe",
  "displayName": null,
  "homeDrive": null,
  "passwordNotRequired": false,
  "createTimeStamp": "11/27/2014 8:14:52 PM",
  "uSNChanged": "47248",
  "sAMAccountName": "sample",
  "userAccountControl": 512,
  "homePhone": null,
  "postalCode": null
}
```

7. Test that you can update a user object on the Active Directory server by sending a PUT request with the complete object, including the user ID in the URL.

   The following request updates user `Jane Doe`'s entry, including her ID in the request. The update sends the same information that was sent in the `create` request, but adds an `employeeNumber`:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0"  `
--header "Content-Type: application/json" `
--header "If-Match: *" `
--request PUT `
--data '{
  \"distinguishedName\" : \"CN=Jane Doe,CN=Users,DC=EXAMPLE,DC=COM\",
  \"sn\" : \"Doe\",
  \"cn\" : \"Jane Doe\",
  \"userPrincipalName\" : \"janedoe@example.com\",
  \"enabled\" : true,
  \"password\" : \"Passw0rd\",
  \"telephoneNumber\" : \"0052-611-091\",
  \"employeeNumber\": \"567893\"
 }' `
"http://localhost:8080/openidm/system/adpowershell/account/42725210-8dce-4fdf-b0e0-393cf0377fdf"
{
  "_id": "42725210-8dce-4fdf-b0e0-393cf0377fdf",
  "title": null,
  "uSNCreated": "47244",
  "pwdLastSet": "130615906375709689",
  "cannotChangePassword": false,
  "telephoneNumber": "0052-611-091",
  "smartcardLogonRequired": false,
  "badPwdCount": "0",
  "department": null,
  "distinguishedName": "CN=Jane Doe,CN=Users,DC=example,DC=com",
  "badPasswordTime": "0",
  "employeeID": null,
  "cn": "Jane Doe",
  "division": null,
  "description": null,
  "userPrincipalName": "janedoe@example.com",
  "passwordNeverExpires": false,
  "company": null,
  "memberOf": [],
  "givenName": null,
  "streetAddress": null,
  "sn": "Doe",
  "initials": null,
  "logonCount": "0",
  "homeDirectory": null,
  "employeeNumber": "567893",
  "objectGUID": "42725210-8dce-4fdf-b0e0-393cf0377fdf",
  "manager": null,
  "lastLogon": "0",
  "trustedForDelegation": false,
  "scriptPath": null,
  "allowReversiblePasswordEncryption": false,
  "modifyTimeStamp": "11/27/2014 8:37:17 PM",
  "whenCreated": "11/27/2014 8:14:52 PM",
  "whenChanged": "11/27/2014 8:37:17 PM",
```

```
  "accountExpirationDate": null,
  "name": "Jane Doe",
  "displayName": null,
  "homeDrive": null,
  "passwordNotRequired": false,
  "createTimeStamp": "11/27/2014 8:14:52 PM",
  "uSNChanged": "47253",
  "sAMAccountName": "sample",
  "userAccountControl": 512,
  "homePhone": null,
  "postalCode": null
}
```

8. Test whether you are able to delete a user object on the Active Directory server by sending a DELETE request with the user ID in the URL.

   The following request deletes user `Jane Doe`'s entry:

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0"  `
--request DELETE `
"http://localhost:8080/openidm/system/adpowershell/account/42725210-8dce-4fdf-b0e0-393cf0377fdf"
```

   The response includes the complete user object that was deleted.

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

```
curl.exe `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0"  `
--request GET `
"http://localhost:8080/openidm/system/adpowershell/account/42725210-8dce-4fdf-b0e0-393cf0377fdf"
{
  "message": "",
  "reason": "Not Found",
  "code": 404
}
```

Synchronize data between IDM and Azure Active Directory

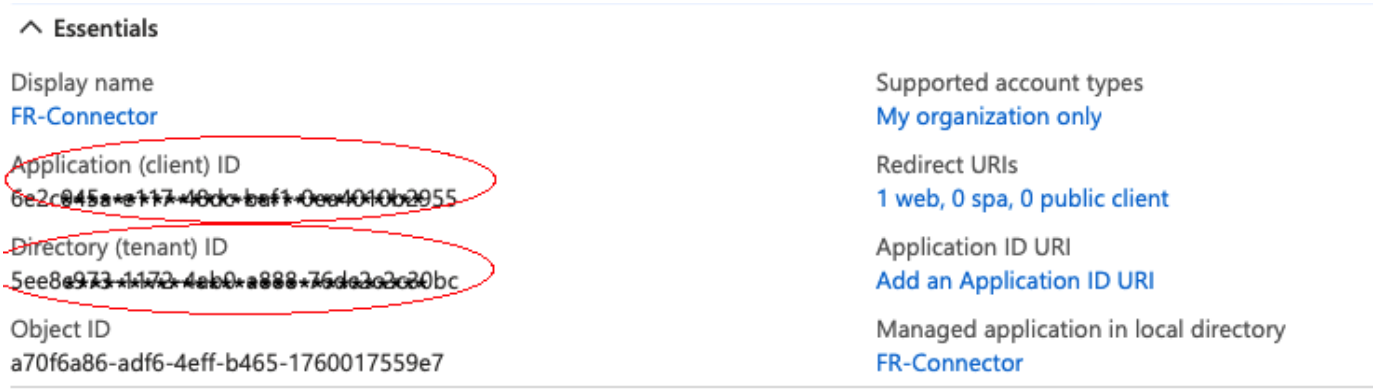
Azure Active Directory (Azure AD) is Microsoft's cloud-based identity and access management service that lets users sign in and access resources. This sample uses the Microsoft Graph API connector to synchronize IDM managed users and Azure AD users.

Prepare the sample

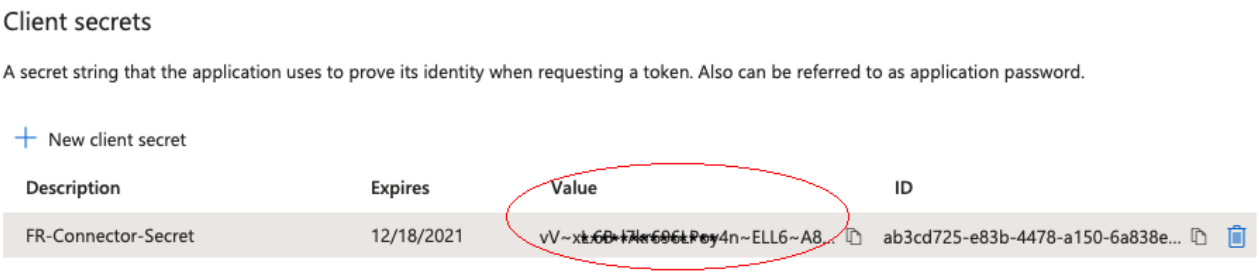
Before you can run this sample, you must register an application with Azure. You need a Microsoft Azure subscription to complete this procedure:

1. Log in to the MS Azure portal as an administrative user.
2. Under **Azure services** , select **App registrations**.
3. On the **Register an application** page, enter a name for the application; for example, **FR-Connector**.
4. Select the supported account types, and enter a **Redirect URI**.

   The redirect URI is the IDM URI that Azure should redirect to after successful authentication; for example, `https://idm.example.com:8443/` .
5. On the new registration page for your application, make a note of the **Application (client) ID** and the **Directory (tenant) ID**. You will need these to configure the connector:



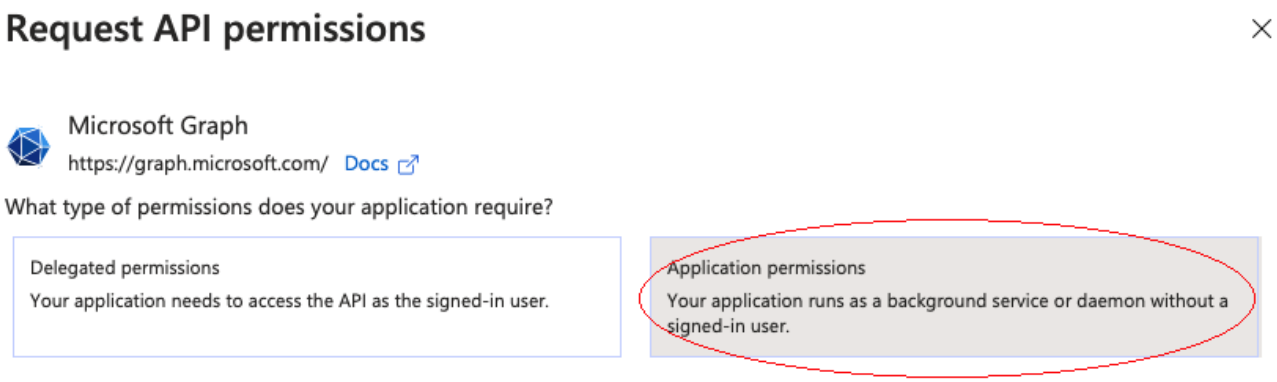
6. Generate a client secret:
   1. Select **Certificates & secrets** > **New client secret** .
   2. Enter a description, select an expiration date, and click **Add**.
   3. Copy the client secret **Value**:



> **Important**
>
> You will not be able to retrieve the client secret in cleartext after you exit this screen.

7. Set the API permissions:

    1. Select **API permissions**, click **Microsoft Graph**, and then click **Application permissions**.



    2. From the **User** item, select the following permissions:

        - `User.Export.All`
        - `User.ManageIdentities.All`
        - `User.Read.All`
        - `User.ReadWrite.All`

    3. From the **Group** item, select the following permissions:

        - `Group.Create`
        - `Group.Read.All`
        - `Group.ReadWrite.All`

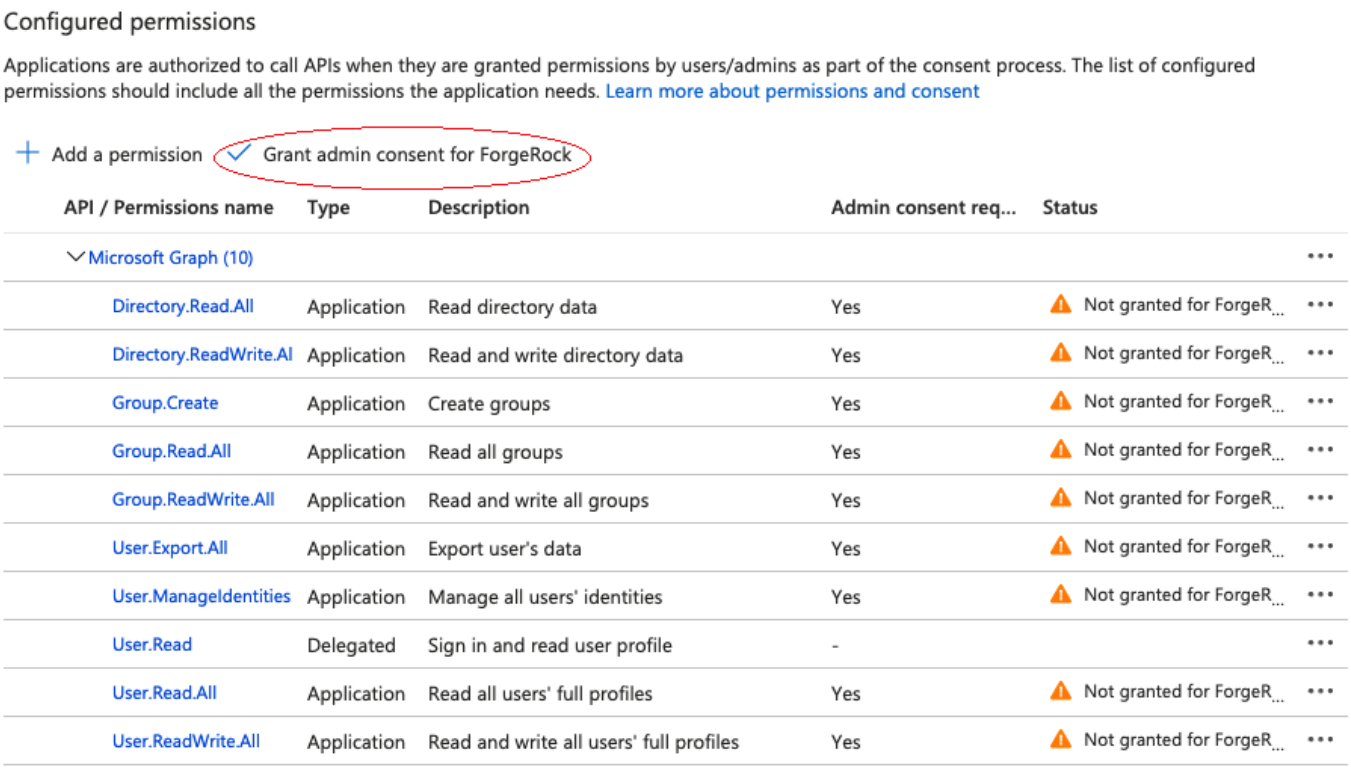
    4. From the **Directory** item, select the following permissions:

        - `Directory.Read.All`
        - `Directory.ReadWrite.All`

    5. Click **Add permissions** .

8. Grant admin consent for the API permissions:

    On the **Configured permissions** page, **Grant admin consent for org-name**, then click **Yes**.



Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. Learn more about permissions and consent

+ Add a permission ✓ Grant admin consent for ForgeRock

API / Permissions name	Type	Description	Admin consent req...	Status
Microsoft Graph (10)				
Directory.Read.All	Application	Read directory data	Yes	Not granted for ForgeR...
Directory.ReadWrite.Al	Application	Read and write directory data	Yes	Not granted for ForgeR...
Group.Create	Application	Create groups	Yes	Not granted for ForgeR...
Group.Read.All	Application	Read all groups	Yes	Not granted for ForgeR...
Group.ReadWrite.All	Application	Read and write all groups	Yes	Not granted for ForgeR...
User.Export.All	Application	Export user's data	Yes	Not granted for ForgeR...
User.ManageIdentities	Application	Manage all users' identities	Yes	Not granted for ForgeR...
User.Read	Delegated	Sign in and read user profile	-	
User.Read.All	Application	Read all users' full profiles	Yes	Not granted for ForgeR...
User.ReadWrite.All	Application	Read and write all users' full profiles	Yes	Not granted for ForgeR...

Configure the MS Graph API connector

This procedure uses the admin UI to configure the connector. You can also edit the `samples/sync-with-azuread/conf/provisioner.openicf-azuread.json` file directly.

1. Start IDM with the configuration for the AzureAD sample:

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

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

   This URL reflects the host on which IDM is installed, and must be the same as the `Redirect URI` that you set when you registered your Azure application.

3. Select **Configure > Connectors**, and click the Azuread connector.

4. Under **General Details**, select **Enabled**.

5. Under **Base Connector Details**, enter at least the **Tenant**, **ClientID**, and **Client Secret** that you obtained when you prepared the sample, and then click **Save**.

   After you click **Save**, IDM validates the connector configuration. If you do not see an error, your connector is configured correctly.

Run the sample

This procedure uses create, read, update, and delete (CRUD) operations on the Azure AD resource, to verify that the connector is working as expected. The procedure uses a combination of REST commands, to manage users and groups in Azure AD, and the admin UI, to manage IDM users and reconcile objects between the Azure AD and IDM.

The sample configuration has two mappings: one *from* Azure AD *to* the managed user repository, and one *from* the managed user repository *to* the users in Azure AD.

Before you can synchronize accounts between the two data stores, you must update the second mapping with your tenant name:

1. In the admin UI, select **Configure > Mappings**.
2. Click on mapping 2 (**managedUser_systemAzureadUser**).
3. On the **Properties** tab, under the **Attributes** grid, click the `userName` **to** `userPrincipalName` mapping.
4. On the **Transformation Script** tab, replace `<my tenant>` with the name of your tenant. For example:

```
source +'@example.onmicrosoft.com'
```

5. Click **Save**.

> **Note**
>
> All of the commands shown here assume that your domain is `example.com` . Adjust the examples to match your domain.

Manage users in Azure AD

1. Create a user entry in Azure AD, over REST. This command creates an entry for user `Sam Carter` :

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--header "content-type: application/json" \
--data '{
  "surname": "Carter",
  "displayName": "Sam Carter",
  "givenName": "Sam",
  "userType": "Member",
  "accountEnabled": true,
  "mail": "scarter@example.com",
  "country": "US",
  "mailNickname": "scarter",
  "userPrincipalName": "scarter@example.onmicrosoft.com",
  "__PASSWORD__": "MyPassw0rd"
}' \
"http://localhost:8080/openidm/system/azuread/user?_action=create"
{
  "_id": "be14f228-1d3a-4f31-aeee-ba6e8419b1b0",
  "accountEnabled": true,
  "memberOf": [],
  "userPrincipalName": "scarter@example.onmicrosoft.com",
  "mailNickname": "scarter",
  "givenName": "Sam",
  "proxyAddresses": [
    "SMTP:scarter@example.com"
  ],
  "createdDateTime": "2021-03-31T13:47:59Z",
  "onPremisesExtensionAttributes": {
    ...
  },
  "surname": "Carter",
  "imAddresses": [],
  "userType": "Member",
  "manager": null,
  "country": "US",
  "licenses": [],
  "id": "be14f228-1d3a-4f31-aeee-ba6e8419b1b0",
  "mail": "scarter@example.com",
  "displayName": "Sam Carter",
  "identities": [
    {
      "signInType": "userPrincipalName",
      "issuerAssignedId": "scarter@example.onmicrosoft.com",
      "issuer": "example.onmicrosoft.com"
    }
  ],
  "__NAME__": "scarter@example.onmicrosoft.com",
  "businessPhones": []
}
```

> **Note**
>
> Take note of the ID of the new user (`be14f228-1d3a-4f31-aeee-ba6e8419b1b0` in this example). You will need this ID for additional commands in this example.

2. Reconcile the Azure AD resource with the managed user repository.

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

   1. In the admin UI, select **Configure > Mappings**.
   2. On mapping 1 (**systemAzureadUser_managedUser**), click **Reconcile**.
   3. Select **Manage > User**, and verify that the user Sam Carter exists in the repository.

3. Update Sam Carter's `country` property in IDM:

   1. Select **Manage > User**, and click **Sam Carter**'s entry.
   2. Change his **Country** property from US to FR, and click **Save**.

   As a result of implicit synchronization, Sam Carter's country should be updated automatically in the Azure AD resource.

4. Read the value of Sam Carter's `country` attribute in your Azure AD, specifying the ID you retrieved when you created the user:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/azuread/user/be14f228-1d3a-4f31-aeee-ba6e8419b1b0?
_fields=country"
{
  "_id": "be14f228-1d3a-4f31-aeee-ba6e8419b1b0",
  "country": "FR"
}
```

Manage groups in Azure AD

1. Create a basic group entry in Azure AD:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "description": "Group used for Azure AD sample.",
  "displayName": "AzureAD Test Group",
  "mailNickname": "ExampleTestGroup",
  "mailEnabled": false,
  "securityEnabled": true
}' \
"http://localhost:8080/openidm/system/azuread/group?_action=create"
{
  "_id": "c628713f-e6c8-43a4-8c5d-9c9ee437d950",
  "description": "Group used for Azure AD sample.",
  "mailNickname": "ExampleTestGroup",
  "groupTypes": [],
  "displayName": "AzureAD Test Group",
  "securityIdentifier": "S-1-12-1-3324539199-1134880456-2661047692-1356412900",
  "proxyAddresses": [],
  "mailEnabled": false,
  "createdDateTime": "2021-04-01T12:40:22Z",
  "securityEnabled": true,
  "members": [],
  "__NAME__": "AzureAD Test Group",
  "creationOptions": []
}
```

2. Add Sam Carter to the `AzureAD Test Group` that you have just created. Choose one of the following methods:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request PUT \
--header "If-Match:*" \
--header "content-type: application/json" \
--data '{
  "memberOf": ["c628713f-e6c8-43a4-8c5d-9c9ee437d950"]
}' \
"http://localhost:8080/openidm/system/azuread/user/be14f228-1d3a-4f31-aeee-ba6e8419b1b0"
{
  "_id": "be14f228-1d3a-4f31-aeee-ba6e8419b1b0",
  "manager": null,
  "userPrincipalName": "scarter@example.onmicrosoft.com",
  "userType": "Member",
  "country": "FR",
  "createdDateTime": "2021-03-31T13:47:59Z",
  "givenName": "Sam",
  "__NAME__": "scarter@example.onmicrosoft.com",
  "onPremisesExtensionAttributes": {
    ...
  },
  "mailNickname": "scarter",
  "licenses": [],
  "businessPhones": [],
  "displayName": "Sam Carter",
  "imAddresses": [],
  "id": "be14f228-1d3a-4f31-aeee-ba6e8419b1b0",
  "mail": "scarter@example.com",
  "proxyAddresses": [
    "smtp:scarter@example.onmicrosoft.com",
    "SMTP:scarter@example.com"
  ],
  "accountEnabled": true,
  "memberOf": [
    "c628713f-e6c8-43a4-8c5d-9c9ee437d950"
  ],
  "identities": [
    {
      "signInType": "userPrincipalName",
      "issuerAssignedId": "scarter@example.onmicrosoft.com",
      "issuer": "example.onmicrosoft.com"
    }
  ],
  "surname": "Carter"
}
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PUT \
--header "If-Match:*" \
--data '{
  "members": ["be14f228-1d3a-4f31-aeee-ba6e8419b1b0"]
}' \
"http://localhost:8080/openidm/system/azuread/group/c628713f-e6c8-43a4-8c5d-9c9ee437d950"
{
  "_id": "c628713f-e6c8-43a4-8c5d-9c9ee437d950",
  "description": "Group used for Azure AD sample.",
  "mailNickname": "ExampleTestGroup",
  "groupTypes": [],
  "displayName": "AzureAD Test Group",
  "securityIdentifier": "S-1-12-1-3324539199-1134880456-2661047692-1356412900",
  "proxyAddresses": [],
  "mailEnabled": false,
  "createdDateTime": "2021-04-01T12:40:22Z",
  "securityEnabled": true,
  "members": [
    "be14f228-1d3a-4f31-aeee-ba6e8419b1b0"
  ],
  "__NAME__": "AzureAD Test Group",
  "creationOptions": []
}
```

3. Read the group entry's `members` property to verify that the Sam Carter has been added:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/azuread/group/c628713f-e6c8-43a4-8c5d-9c9ee437d950?
_fields=members"
{
  "_id": "c628713f-e6c8-43a4-8c5d-9c9ee437d950",
  "members": [
    "be14f228-1d3a-4f31-aeee-ba6e8419b1b0"
  ]
}
```

4. Delete the group entry:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/system/azuread/group/c628713f-e6c8-43a4-8c5d-9c9ee437d950"
{
  "_id": "c628713f-e6c8-43a4-8c5d-9c9ee437d950",
  "description": "Group used for Azure AD sample.",
  "mailNickname": "ExampleTestGroup",
  "groupTypes": [],
  "displayName": "AzureAD Test Group",
  "securityIdentifier": "S-1-12-1-3324539199-1134880456-2661047692-1356412900",
  "proxyAddresses": [],
  "mailEnabled": false,
  "createdDateTime": "2021-04-01T12:40:22Z",
  "securityEnabled": true,
  "members": [
    "be14f228-1d3a-4f31-aeee-ba6e8419b1b0"
  ],
  "__NAME__": "AzureAD Test Group",
  "creationOptions": []
}
```

5. Delete user Sam Carter, to return your Azure AD resource to its original state:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/system/azuread/user/be14f228-1d3a-4f31-aeee-ba6e8419b1b0"
{
  "_id": "be14f228-1d3a-4f31-aeee-ba6e8419b1b0",
  "manager": null,
  "userPrincipalName": "scarter@example.onmicrosoft.com",
  "userType": "Member",
  "country": "FR",
  "createdDateTime": "2021-03-31T13:47:59Z",
  "givenName": "Sam",
  "__NAME__": "scarter@example.onmicrosoft.com",
  "onPremisesExtensionAttributes": {
    ...
  },
  "mailNickname": "scarter",
  "licenses": [],
  "businessPhones": [],
  "displayName": "Sam Carter",
  "imAddresses": [],
  "id": "be14f228-1d3a-4f31-aeee-ba6e8419b1b0",
  "mail": "scarter@example.com",
  "proxyAddresses": [
    "smtp:scarter@example.onmicrosoft.com",
    "SMTP:scarter@example.com"
  ],
  "accountEnabled": true,
  "memberOf": [],
  "identities": [
    {
      "signInType": "userPrincipalName",
      "issuerAssignedId": "scarter@example.onmicrosoft.com",
      "issuer": "example.onmicrosoft.com"
    }
  ],
  "surname": "Carter"
}
```

In this sample, you used the MS Graph API connector to add and delete user and group objects in your Azure AD tenant and to reconcile users between Azure AD and IDM. You can expand on this sample by customizing the connector configuration to provide additional synchronization functionality between IDM and Azure AD. For information about configuring connectors, refer to Connector reference.

Connect to a MySQL database with ScriptedSQL

This sample uses the Groovy Connector Toolkit bundled with IDM (`openidm/connectors/groovy-connector-1.5.20.14.jar`) to implement a ScriptedSQL connector that interacts with an external MySQL database (HRDB), and also demonstrates the following functionality:

- Complex data types.

  Complex data types can be stored, retrieved and synchronized like any other object property. They are stored in the managed data as JSON objects, represented as a string, but can be mapped to external resources in any format required. You can customize the mapping to do additional work with or transformations on the complex data types.

  This sample defines one complex data type, `cars` , discussed in more detail later in this section.

- Event hooks to perform actions.

  The mapping from the internal repository to the external `hrdb` database includes two script hooks. The first hook is for an `onCreate` event and the second is for an `onUpdate` event.

- Custom scripted endpoints.

  Custom scripted endpoints are configured in the provisioner configuration file and allow you to execute custom scripts over REST. This sample uses a custom scripted endpoint to reset the database and populate it with data.

> **Caution**
>
> Because MySQL cannot "un-hash" user passwords there is no way for a reconciliation operation to retrieve the password from MySQL and store it in the managed user object. This issue might impact configurations that support multiple external resources in that passwords might not be synchronized immediately after reconciliation from MySQL to the managed user repository. Users who are missing in the repository will be created by reconciliation but their passwords will be empty. When those users are synchronized to other external resources, they will have empty passwords in those resources. Additional scripting might be required to handle this situation, depending on the requirements of your deployment.

The Groovy scripts required for the sample are located in the `samples/scripted-sql-with-mysql/tools` directory. Note that the power of the Groovy connector is in the associated Groovy scripts, and their application in your particular deployment. The scripts provided with this sample are specific to the sample. You must customize these scripts to address the requirements of your specific deployment. The sample scripts are a good starting point on which to base your customization.

Configure the external MySQL database

This sample assumes a database running on the localhost.

1. Download MySQL Connector/J version 8.0 or later.
2. Unpack the downloaded file, and copy the .jar file to `openidm/lib` :

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

> **Note**
>
> If you are running this sample with an SQL database other than MySQL, download the corresponding driver and place it in the `openidm/lib` directory. It is not necessary to create an OSGi bundle for the driver.

3. Set up MySQL to listen on localhost, port `3306`. IDM will connect to the `hrdb` database as user `root` with password `password`.

   To use an existing MySQL instance that runs on a different host or port, or to change the database credentials, edit the `configurationProperties` in the connector configuration file (`samples/scripted-sql-with-mysql/conf/provisioner.openicf-hrdb.json`) before you start the sample. The default configuration is as follows:

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

   > **Note**
   >
   > The default configuration expects SSL, *which is strongly advised in a production environment*. If you are running this in a test environment, you can bypass the SSL requirement:
   >
   > - Add `&useSSL=false` to the end of the `url`.
   > - If you are running MySQL 8.0.11+, add `&allowPublicKeyRetrieval=true` to the end of the `url`.

4. Set up the `hrdb` database, with which IDM will synchronize its managed user repository:

   ```
   mysql -u root -p
   Enter password:
   Welcome to the MySQL monitor.  Commands end with ; or \g.
   Your MySQL connection id is 3
   Server version: 5.7.13 MySQL Community Server (GPL)

   Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

   Oracle is a registered trademark of Oracle Corporation and/or its
   affiliates. Other names may be trademarks of their respective
   owners.

   Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

   mysql> CREATE DATABASE hrdb CHARACTER SET utf8 COLLATE utf8_bin;
   Query OK, 1 row affected (0.00 sec)
   ```

5. Configure your GRANT permissions:

   ```
   CREATE USER IF NOT EXISTS 'root'@'%' IDENTIFIED BY 'password';
   GRANT ALL PRIVILEGES ON hrdb.* TO 'root'@'%' WITH GRANT OPTION;
   ```

Run the sample

The mapping configuration file (`sync.json`) for this sample includes the mapping `systemHrdb_managedUser` . You will use this mapping to synchronize users from the source `hrdb` database with the target IDM repository.

1. Start IDM with the configuration for the ScriptedSQL sample:

```
/path/to/openidm/startup.sh -p samples/scripted-sql-with-mysql
```

2. Run the custom script (`samples/scripted-sql-with-mysql/tools/ResetDatabaseScript.groovy`) to reset the database and populate it with sample data.

> **Tip**
>
> You can run the script again, at any point, to reset the database.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/hrdb?_action=script&scriptId=ResetDatabase"
{
  "actions": [
    {
      "result": "Database reset successful."
    }
  ]
}
```

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

3. Review the contents of the database:

```
mysql -u root -p
Enter password:
mysql > use hrdb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql > select * from users;
-----------------------------------------------------------------------...
| id | uid    | password     | firstname | lastname | fullname      | email  ...
+-----------------------------------------------------------------------...
|  1 | bob    | e38ad2149... | Bob       | Fleming  | Bob Fleming   | Bob.Fle...
|  2 | rowley | 2aa60a8ff... | Rowley    | Birkin   | Rowley Birkin | Rowley....
|  3 | louis  | 1119cfd37... | Louis     | Balfour  | Louis Balfour | Louis.B...
|  4 | john   | a1d7584da... | John      | Smith    | John Smith    | John.Sm...
|  5 | jdoe   | edba955d0... | John      | Doe      | John Doe      | John.Do...
+-----------------------------------------------------------------+-------...
5 rows in set (0.00 sec)
```

> **Note**
>
> The passwords in the above output are SHA-1 hashed because they cannot be read into IDM as cleartext. The SHA-1 Hash function is used for compatibility reasons. Use a more secure algorithm in a production database.

4. Reconcile the hrdb database with the managed user repository.

    1. To reconcile the repository by using the Administration UI:

        1. Log in to the admin UI at the URL `https://localhost:8443/admin` as the default administrative user (`openidm-admin`) with password `openidm-admin`.
        2. Select **Configure > Mappings**.

           The Mappings page shows two mappings, one from the `hrdb` database to the IDM managed user repository (`managed/user`), and one in the opposite direction.
        3. Click **Reconcile** on the first mapping (systemHrdb_managedUser).

    2. To reconcile the repository by using the command-line, launch the reconciliation operation with the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemHrdb_managedUser&waitForCompletion=true"
{
  "state": "SUCCESS",
  "_id": "f3c618aa-cc3b-49ed-9a3a-00b012db2513"
}
```

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

5. Retrieve the list of users from the repository.

    1. To retrieve the users in the repository from the admin UI:

        1. Select **Manage > User** to display the *User List*.

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

        2. To retrieve the details of a specific user, click that user entry.

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryId=query-all-ids"
{
  "result": [
    {
      "_id": "1b379e4d-3b8d-47e7-93d5-a72c4b483e39",
      "_rev": "000000002d93e471"
    },
    {
      "_id": "a658f751-d6e9-4b5d-af56-071a9b05c3af",
      "_rev": "000000003c83d48a"
    },
    {
      "_id": "5b31027b-09f8-4c7f-abfa-c6bc86ae3943",
      "_rev": "00000000b042e559"
    },
    {
      "_id": "1b3f6b06-1752-4c40-ba34-51d30b184b9d",
      "_rev": "0000000092bdda6d"
    },
    {
      "_id": "9c62f0d2-47e2-4fc5-89d1-b50b782b1022",
      "_rev": "0000000025cdd3c6"
    }
  ],
  "resultCount": 5,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

To retrieve a complete user record, query the userName of the individual user entry. The following query returns the record for the user `Rowley Birkin`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/?_queryId=for-userName&uid=rowley"
"result": [
    {
      "_id": "1b379e4d-3b8d-47e7-93d5-a72c4b483e39",
      "_rev": "000000002d93e471",
      "mail": "Rowley.Birkin@example.com",
      "userName": "rowley",
      "sn": "Birkin",
      "organization": "SALES",
      "givenName": "Rowley",
      "cars": [
        {
          "year": "2013",
          "make": "BMW",
          "model": "328ci"
        },
        {
          "year": "2010",
          "make": "Lexus",
          "model": "ES300"
        }
      ],
      "accountStatus": "active",
      "effectiveRoles": [],
      "effectiveAssignments": []
    }
  ],
  ...
}
```

Regardless of how you have retrieved Rowley Birkin's entry, note the `cars` property in this user's entry. This property demonstrates a complex object, stored in JSON format in the user entry, as a list that contains multiple objects. In the MySQL database, the `car` table joins to the `users` table through a `cars.users_id` column. The Groovy scripts read this data from MySQL and repackage it in a way that IDM can understand. With support for complex objects, the data is passed through to IDM as a list of `car` objects. Data is synchronized from IDM to MySQL in the same way. Complex objects can also be nested to any depth.

Group membership (not demonstrated here) is maintained with a traditional "join table" in MySQL (`groups_users`). IDM does not maintain group membership in this way, so the Groovy scripts do the work to translate membership between the two resources.

Test the event hooks

This sample uses the `onCreate` and `onUpdate` hooks to log messages when a user is created or updated in the external database.

The sample's `conf/sync.json` file defines these event hooks as follows:

```
...
    {
        "name" : "managedUser_systemHrdb",
        "source" : "managed/user",
        "target" : "system/hrdb/account",
        "links" : "systemHrdb_managedUser",
        "correlationQuery" : {
            "type" : "text/javascript",
            "source" : "({'_queryFilter': 'uid eq \"' + source.userName + '\"'});"
        },
        "onCreate" : {
            "type" : "text/javascript",
            "source" : "logger.info(\"Creating new user in external repo\")"
        },
        "onUpdate" : {
            "type" : "text/javascript",
            "source" : "logger.info(\"Updating existing user in external repo\")"
        },
...
```

Using these event hooks, IDM logs a message when a user is created or updated in the external database. In this sample, the script source is included in the mapping. However, a script can also be called from an external file. For more information about event hooks, refer to Script triggers.

To test the event hooks, create a new managed user as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
    "mail":"fdoe@example.com",
    "sn":"Doe",
    "telephoneNumber":"555-1234",
    "userName":"fdoe",
    "givenName":"Felicitas",
    "description":"Felicitas Doe",
    "displayName":"fdoe"}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "_id": "2e5e9748-77e6-4019-90e1-abe9ab897343",
  "_rev": "0000000015b2d4ba",
  "mail": "fdoe@example.com",
  "sn": "Doe",
  "telephoneNumber": "555-1234",
  "userName": "fdoe",
  "givenName": "Felicitas",
  "description": "Felicitas Doe",
  "displayName": "fdoe",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

The implicit synchronization between the managed user repository and the HRDB database creates that user in the database automatically.

Check the latest log file at `path/to/openidm/logs/openidm0.log.0`. You should see the following message at the end of the log:

```
INFO: Creating new user in external repo
```

Query the new user entry in the HRDB database:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/hrdb/account?_queryFilter=uid+eq+'fdoe'"
{
  "result": [
    {
      "_id": "6",
      "cars": [],
      "firstName": "Felicitas",
      "uid": "fdoe",
      "lastName": "Doe",
      "organization": "IDM",
      "fullName": "Felicitas Doe",
      "email": "fdoe@example.com"
    }
  ],
  …
}
```

Update fdoe's entry in the HRDB database with a patch request. The following request updates the user's `organization` field:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
   "operation" : "replace",
   "field" : "organization",
   "value" : "example.com"
} ]' \
"http://localhost:8080/openidm/system/hrdb/account/6"
{
  "_id": "6",
  "cars": [],
  "firstName": "Felicitas",
  "uid": "fdoe",
  "lastName": "Doe",
  "organization": "example.com",
  "fullName": "Felicitas Doe",
  "email": "fdoe@example.com"
}
```

Note that this update does not reference the `onUpdate` script hook so this change is not logged in `openidm0.log.0`.

Run the sample with paging

All OpenICF connectors from version 1.4 onwards support the use of paging parameters to restrict query results. The following command indicates that only two records should be returned (`_pageSize=2`) and that the records should be sorted according to their `timestamp` and `_id` (`_sortKeys=timestamp,_id`). Including the `timestamp` in the sort ensures that, as you page through the set, changes to records that have already been visited are not lost. Instead, those records are pushed onto the last page:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/hrdb/account?_queryFilter=true&_pageSize=2&_sortKeys=timestamp,_id"
{
  "result": [
    {
      "_id": "1",
      "firstName": "Bob",
      "cars": [
        {
          "year": "1979",
          "make": "Ford",
          "model": "Pinto"
        }
      ],
      "fullName": "Bob Fleming",
      "email": "Bob.Fleming@example.com",
      "uid": "bob",
      "lastName": "Fleming",
      "organization": "HR"
    },
    {
      "_id": "2",
      "firstName": "Rowley",
      "cars": [
        {
          "year": "2013",
          "make": "BMW",
          "model": "328ci"
        }
      ],
      "fullName": "Rowley Birkin",
      "email": "Rowley.Birkin@example.com",
      "uid": "rowley",
      "lastName": "Birkin",
      "organization": "SALES"
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": "2018-04-05+16%3A30%3A22.0%2C2",
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

The `pagedResultsCookie` is used by the server to keep track of the position in the search results. You can ignore the `"remainingPagedResults": -1` in the output. The real value of this property is not returned because the scripts that the connector uses do not do any counting of the records in the resource.

Using the `pagedResultsCookie` from the previous step, run a similar query, to retrieve the following set of records in the database:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/hrdb/account?_queryId=query-all-
ids&_pageSize=2&_sortKeys=timestamp,_id&_pagedResultsCookie=2018-04-05+16%3A30%3A22.0%2C2"
{
  "result": [
    {
      "_id": "3",
    },
    {
      "_id": "4",
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": "2018-04-05+16%3A30%3A22.0%2C4",
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

For more information about paging, refer to Page Query Results.

Direct audit information to MySQL

The sample includes an external CSV file and a mapping between objects in that file and the managed user repository. The reconciliations across this mapping generate the audit records that will be directed to the MySQL database. The connection to the MySQL database is through a ScriptedSQL implementation of the Groovy Connector Toolkit.

About the configuration files

The files that demonstrate the functionality of this sample are located under `/path/to/openidm/samples/audit-jdbc/`, in the `conf/` and `data/` directories.

The following files play important roles in this sample:

`conf/provisioner.openicf-auditdb.json`

This file provides the configuration for the Scripted SQL implementation of the Groovy Connector. The file specifies, among other things, the connection details to the MySQL database, the connector version information, and the object types that are supported for this connection. For more information, refer to Groovy Connector Toolkit.

`conf/provisioner.openicf-csvfile.json`

This file provides the configuration for this instance of the CSV connector. It includes, among other things, the location of the CSV file resource.

`conf/sync.json`

Provides the mapping between managed users and the data set in the CSV file.

`conf/audit.json`

This file configures the router as the audit event handler, and routes audit logs to a remote system, identified as `auditdb`.

`data/csvConnectorData.csv`

This file contains the sample data set that will be reconciled to the managed user repository.

`data/sample_audit_db.mysql`

This file sets up the schema for the MySQL database that will contain the audit logs.

`tools/*.groovy`

The Groovy scripts in this directory allow the connector to perform operations on the MySQL database.

Configure the MySQL database

The sample assumes the following MySQL configuration:

- The database is available on the local host.
- The database listens on the standard MySQL port, 3306.
- You can connect to the MySQL database, as user `root` with password `password`.

Before you start this sample, MySQL must be installed and running, and must include the database required for the sample. In addition, IDM must include the connector JAR required to connect to the MySQL database.

1. Install and configure MySQL.
2. This step sets up an `audit` database with tables that correspond to the various audit events. When MySQL is up and running, import the database schema to set up the database required for the sample:

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

3. To view the tables in the audit database, use the following command:

```
mysql -u root -p
Enter password:password
mysql> use audit
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+---------------------+
| Tables_in_audit     |
+---------------------+
| auditaccess         |
| auditactivity       |
| auditauthentication |
| auditconfig         |
| auditrecon          |
| auditsync           |
+---------------------+
6 rows in set (0.00 sec)
```

4. Download MySQL Connector/J, version 8.0 or later from the MySQL website. Unpack the delivery, and copy the .jar into the `openidm/bundle` directory:

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

5. Edit the `url` property in the SQL connector configuration file (`openidm/samples/audit-jdbc/conf/provisioner.openicf-auditdb.json`) to match the host and port of your MySQL instance. The default configuration is as follows:

```
"url" : "jdbc:mysql://localhost:3306/audit?serverTimezone=UTC",
```

> **Note**
>
> The default configuration expects SSL, *which is strongly advised in a production environment*. If you are running this in a test environment, you can bypass the SSL requirement:
>
> - Add `&useSSL=false` to the end of the `url`.
> - If you are running MySQL 8.0.11+, add `&allowPublicKeyRetrieval=true` to the end of the `url`.

Run the sample

In this section, you will start IDM, then run a reconciliation between the CSV file and the managed user repository. After the reconciliation, you should be able to read the audit logs in the `audit` database on your MySQL instance.

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

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

2. Reconcile the two data sources.

   - To run the reconciliation over REST, use the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemCsvfileAccounts_managedUser&waitForCompletion=true"
{
  "_id": "a3664c26-bf82-4100-b411-19edc248c306-7",
  "state": "SUCCESS"
}
```

   - To run the reconciliation from the admin UI, select **Configure > Mappings**, select the `systemCsvfileAccounts_managedUser` mapping, and then select **Reconcile**.

3. Inspect the tables in the `audit` database to see how the logs have been routed to that location.

   The following example displays the reconciliation audit logs:

```
mysql -u root -p
Enter password:password...
mysql> use audit;...
mysql> show tables;
+---------------------+
+---------------------+
| Tables_in_audit     |
+---------------------+
| auditaccess         |
| auditactivity       |
| auditauthentication |
| auditconfig         |
| auditrecon          |
| auditsync           |
+---------------------+
6 rows in set (0.00 sec)
mysql> select * from auditactivity;

+----+-------------+--------------------------+-------------+----------------+---------------+...+
| id | objectid    | activitydate             | eventname   | transactionid  | userid        |...|
+----+-------------+--------------------------+-------------+----------------+---------------+...+
|  1 | 9927b8db*   | 2021-01-25T12:53:00.800Z | activity    | 9927b8db*      | openidm-admin |...|
```

   You can inspect the other audit logs in the same way.

4. By default, the audit configuration in this sample uses the router audit handler for queries, as indicated in the following line from the `conf/audit.json` file:

```
 "handlerForQueries" : "router",
```

With this configuration, when you query the audit logs over REST, the audit data is returned from the router handler (in this case the MySQL database). The following example shows how to query the activity audit log:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/audit/activity?_queryFilter=true"
{
  "result": [
    {
      "_id": "9927b8db-4537-467f-a077-dbe8cab2a4c8-1187",
      "timestamp": "2021-01-25T12:53:00.800Z",
      "userId": "openidm-admin",
      "operation": "CREATE",
      "changedFields": null,
      "objectId": "managed/user/47527af8-f7d5-4b4b-9d8e-af45169016d4",
      "eventName": "activity",
      "trackingIds": null,
      "transactionId": "9927b8db-4537-467f-a077-dbe8cab2a4c8-1109",
      "runAs": "openidm-admin",
      "passwordChanged": false,
      "message": "create",
      "status": "SUCCESS"
    },
  ],
 ...
}
```

You can query the other audit logs in the same way.

Direct audit information to a JMS broker

This sample shows how to configure a Java Message Service (JMS) audit event handler to direct audit information to a JMS broker.

JMS is an API that supports Java-based peer-to-peer messages between clients. The JMS API can create, send, receive, and read messages, reliably and asynchronously. You can set up a JMS audit event handler to publish messages that comply with the Java Message Service Specification Final Release 1.1.

This sample demonstrates the use of the JMS audit event handler. In the sample you will set up communication between IDM and an external JMS Message Broker, as well as Apache ActiveMQ Artemis as the JMS provider and message broker.

> **Note**
>
> JMS topics are not related to ForgeRock audit event topics. The ForgeRock implementation of JMS topics uses the publish/subscribe messaging domain to direct messages to the JMS audit event handler. In contrast, ForgeRock audit event topics specify categories of events.

Dependencies for JMS messaging

The JMS audit event handler requires Apache ActiveMQ Artemis and additional dependencies bundled with the ActiveMQ Artemis delivery. This section lists the dependencies, and where they must be installed in the IDM instance. If you use a different ActiveMQ version, you may need to download the corresponding dependencies separately.

1. Download the following files:
   - Apache ActiveMQ Artemis.

     > **Note**
     >
     > This sample was tested with version 2.20.0.

   - The most recent `bnd` JAR file from https://repo1.maven.org/maven2/biz/aQute/bnd/biz.aQute.bnd/.

     > **Tip**
     >
     > The bnd utility lets you create OSGi bundles for libraries that do not support OSGi.

2. Unpack the ActiveMQ Artemis archive. For example:

```
tar -zxvf ~/Downloads/apache-artemis-2.20.0-bin.tar.gz
```

3. Create a temporary directory, and then change to that directory:

```
mkdir ~/Downloads/tmp
cd ~/Downloads/tmp/
```

4. Move the ActiveMQ Artemis Client and `bnd` JAR files to the temporary directory.

```
mv ~/Downloads/apache-artemis-2.20.0/lib/client/artemis-jms-client-all-2.20.0.jar ~/Downloads/tmp/
mv ~/Downloads/biz.aQute.bnd-version.jar ~/Downloads/tmp/
```

5. Create an OSGi bundle:
   1. In a text editor, create a BND file named `activemq.bnd` with the following contents, and save it to the current directory:

```
version=2.20.0
Export-Package: *;version=${version}
Import-Package: !org.apache.log4j.*,!org.apache.log.*,!org.apache.avalon.framework.logger.*,!
org.apache.avalon.framework.logger.*,!org.glassfish.json.*,!org.conscrypt.*,!
org.apache.logging.*,!org.bouncycastle.jsse.*,!org.eclipse.*,!sun.security.*,!reactor.*,!
org.apache.activemq.artemis.shaded.*,!com.aayushatharva.*,!com.github.luben.zstd,!
com.jcraft.jzlib,!com.ning.compress,!com.ning.compress.lzf,!com.ning.compress.lzf.util,!
com.oracle.svm.core.annotate,!lzma.*,!net.jpountz.*,*
Bundle-Name: ActiveMQArtemis :: Client
Bundle-SymbolicName: org.apache.activemq
Bundle-Version: ${version}
```

Your `tmp/` directory should now contain the following files:

```
ls -1 ~/Downloads/tmp/
activemq.bnd
artemis-jms-client-all-2.20.0.jar
biz.aQute.bnd-version.jar
```

2. In the same directory, create the OSGi bundle archive file. For example:

```
java -jar biz.aQute.bnd-version.jar wrap \
--properties activemq.bnd \
--output artemis-jms-client-all-2.20.0-osgi.jar \
artemis-jms-client-all-2.20.0.jar
```

6. Copy the resulting `artemis-jms-client-all-2.20.0-osgi.jar` file to the `openidm/bundle` directory:

```
cp artemis-jms-client-all-2.20.0-osgi.jar /path/to/openidm/bundle/
```

Configure SSL for Apache ActiveMQ Artemis

For information on configuring Apache ActiveMQ Artemis security features, including SSL, refer to the ActiveMQ Artemis Documentation:

- Security
- Configuring the Transport

Configure a secure port for JMS messages

If you configured SSL for ActiveMQ Artemis, edit `/path/to/openidm/samples/audit-jms/conf/audit.json`, and replace the `java.naming.provider.url`:

```
"java.naming.provider.url" : "ssl://localhost:61617?daemon=true"
```

Start the ActiveMQ Artemis broker and IDM

With the appropriate bundles in the `/path/to/openidm/bundle/` directory, you can start the ActiveMQ Artemis message broker, and then start IDM with the configuration for this sample.

> **Important**
>
> For a full list of ActiveMQ Artemis setup options, refer to Using the Server in the *ActiveMQ Artemis Documentation*.

1. Navigate to the directory where you unpacked the ActiveMQ Artemis binary and run the following command to create the Artemis broker:

```
cd ~/Downloads/apache-artemis-2.20.0/bin
./artemis create fr-audit
Creating ActiveMQ Artemis instance at: /path/to/Downloads/apache-artemis-2.20.0/bin/fr-audit
...
```

2. Start the newly created ActiveMQ Artemis broker:

```
./fr-audit/bin/artemis run
```

3. Start IDM with the sample configuration:

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

> **Note**
>
> If you see the following error in the OSGi console, make sure that you have installed all the required dependencies and that you have started the ActiveMQ Artemis broker.
>
> ```
> SEVERE: Unable to create JmsAuditEventHandler 'jms': null
> ```

Configure and use a JMS consumer application

To take advantage of the ActiveMQ Artemis event broker, the JMS audit sample includes a Java consumer in the following directory: `/path/to/openidm/samples/audit-jms/consumer/`

1. Assuming you have Apache Maven installed on the local system, you can compile the sample consumer with the following commands:

```
cd /path/to/openidm/samples/audit-jms/consumer/
mvn clean install
```

When the build process is complete, you'll see a `BUILD SUCCESS` message:

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 22.852 s
[INFO] Finished at: 2017-02-17T17:21:35+02:00
[INFO] Final Memory: 18M/148M
[INFO] ------------------------------------------------------------------------
```

Note

You might see `[WARNING]` messages during the build. As long as the messages end with `BUILD SUCCESS`, you can proceed with the JMS consumer application.

2. When the consumer is compiled, run one of the following commands in the same directory to output audit messages related to IDM actions:

   - If you *haven't* configured ActiveMQ Artemis on a secure port:

     ```
     mvn \
     exec:java \
     -Dexec.mainClass="consumer.src.main.java.SimpleConsumer" \
     -Dexec.args="org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory tcp://localhost:
     61616"
     ```

   - If you've configured ActiveMQ Artemis on a secure port:

     ```
     MAVEN_OPTS="-Djavax.net.ssl.trustStore=/path/to/openidm/security/truststore" \
     mvn \
     exec:java \
     -Dexec.mainClass="consumer.src.main.java.SimpleConsumer" \
     -Dexec.args="org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory ssl://localhost:
     61617?daemon=true"
     ```

Tip

Look for the message `READY, listening for messages.`

3. Try some actions on IDM, either in a different console or in the admin UI. Watch the output in the `SimpleConsumer` console. For example:

```
--------Message Wed 2022.05.11 at 15:47:07.007 PDT--------
{"auditTopic":"authentication","event":{"_id":"1c25a7d3-e04b-4e4f-ba02-
dfca7b2b0dbd-2883","timestamp":"2022-05-11T22:47:06.375Z","eventName":"SESSION","transactionId":"1c25a7d3-
e04b-4e4f-ba02-dfca7b2b0dbd-2880","trackingIds":["d7cd0e66-4d48-4e71-90ec-357c93938c82","1cab8896-9af1-431a-
aee8-45b5696e8e52"],"userId":"openidm-admin","principal":["openidm-admin"],"entries":
[{"moduleId":"JwtSession","result":"SUCCESSFUL","info":{"org.forgerock.authentication.principal":"openidm-
admin"}}],"result":"SUCCESSFUL","provider":null,"method":"JwtSession"},"_topic":"authentication"}
-------------------------------------------------------------
```

Synchronize data between MongoDB and IDM

This sample uses the Groovy Connector Toolkit to implement a scripted connector that interacts with a MongoDB database. You can use the connector to provision MongoDB database users and roles from an IDM managed repository.

The Groovy Connector Toolkit is bundled with IDM in the JAR `openidm/connectors/groovy-connector-1.5.20.14.jar`.

Sample overview

The Groovy scripts required for the sample are bundled within the MongoDB connector. If you want to customize these scripts, you can specify different scripts by adjusting the `scriptRoots` property and script names in `provisioner.openicf-mongodb.json`.

This sample lets you to synchronize from IDM Managed User to an external MongoDB database.

> **Note**
>
> There is currently no way to synchronize passwords from an external MongoDB database to IDM. Because of this, it is recommended that IDM be used for user creation and password management.

While not demonstrated in this sample, the MongoDB connector can also:

- Synchronize from a dedicated store of IDM Managed MongoDB Roles to an external MongoDB database.
- Synchronize from an external MongoDB database to a dedicated IDM store of Managed MongoDB Roles.

Configure the MongoDB database

This sample assumes a MongoDB database, running on the localhost system. Follow these steps to install and configure the MongoDB database:

1. Use the instructions for downloading and installing MongoDB in the MongoDB Manual. For the supported version of MongoDB, refer to MongoDB connector.
2. Set up MongoDB, based on the `configurationProperties` described in MongoDB connector. By default, MongoDB listens on localhost, port 27017. For the purpose of this sample, set up an administrative user of `myUserAdmin` with a password of `Passw0rd` in the `admin` database. Then create a database in MongoDB named `hrdb`.

   > **Note**
   >
   > The MongoDB administrative user must have the `userAdminAnyDatabase` role, or attempts to update users will fail.

   If want to use an existing MongoDB instance that runs on a different host or port, or you want to change the database credentials, adjust the `configurationProperties` in the connector configuration file (`samples/sync-with-mongodb/conf/provisioner.openicf-mongodb.json`) before you start the sample, as described in Configure the MongoDB connector.
3. Set up the MongoDB database, with which IDM will synchronize its managed user repository, by:
   - Enabling authentication, as described in the following MongoDB document: Enable Auth.

- Setting up users and roles, as described in this MongoDB document: Manage Users and Roles.

Run the sample

In this section, you will start IDM with the sample configuration, test the connection to the MongoDB database, and populate the database with sample data.

The mapping configuration file (`sync.json`) for this sample includes one mapping: `managedUser_systemMongodbAccount` . You will use this mapping to synchronize users between the IDM repository and the MongoDB database:

1. Update `samples/sync-with-mongodb/conf/provisioner.openicf-mongodb.json` with the credentials and database information you created when configuring MongoDB. In our example, `database` would be set to `hrdb` , while `user` would be `myUserAdmin` with `userDatabase` set to `admin` .
2. Start IDM with the configuration for the MongoDB sample:

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

3. Create at least one assignment and role to assign roles to users. In this example, we are creating a role to assign read privileges to users. The role created is conditional, and only assigned to active users:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-type: application/json" \
--request POST \
--data '{
   "name" : "MongoDB Read Access",
   "description": "Basic Read Access to HRDB",
   "mapping" : "managedUser_systemMongodbAccount",
   "attributes": [
     {
       "name": "roles",
       "value": [
         {
           "role": "read",
           "db": "hrdb"
         }
       ],
       "assignmentOperation" : "mergeWithTarget",
       "unassignmentOperation" : "removeFromTarget"
     }
   ]
 }' \
"http://localhost:8080/openidm/managed/assignment?_action=create"
{
  "_id": "fb98f4a5-0f4d-4e22-9e17-79c45c11fe20",
  "_rev": "000000005c2da0eb",
  "name": "MongoDB Read Access",
  "description": "Basic Read Access to HRDB",
  "mapping": "managedUser_systemMongodbAccount",
  "attributes": [
    {
      "name": "roles",
      "value": [
        {
          "role": "read",
          "db": "hrdb"
        }
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ]
}
```

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
   "name" : "MongoDB Read Access",
   "description": "Role for accounts with read access in MongoDB.",
   "condition": "/accountStatus eq \"active\"",
   "assignments": [
     {
       "_ref": "managed/assignment/fb98f4a5-0f4d-4e22-9e17-79c45c11fe20",
       "_refResourceCollection": "managed/assignment",
       "_refResourceId": "fb98f4a5-0f4d-4e22-9e17-79c45c11fe20"
     }
   ]
 }' \
"http://localhost:8080/openidm/managed/role?_action=create"
{
  "_id": "5f16238e-39e1-4f8c-8b16-27d39dc64dc3",
  "_rev": "0000000011e566a2",
  "name": "MongoDB Read Access",
  "description": "Role for accounts with read access in MongoDB.",
  "condition": "/accountStatus eq \"active\""
}
```

4. Create new users in IDM. Note that MongoDB requires user name, password, and roles properties to successfully create a user. In this example, the `read` role is assigned to new users automatically.
5. Reconcile the managed user repository with the external MongoDB database.
   - To reconcile the repository using the admin UI:
     1. Log in to the admin UI at the URL `https://localhost:8443/admin` as the default administrative user (`openidm-admin`) with password `openidm-admin`.
     2. Select **Configure > Mappings**.

        The Mappings page shows one mapping: From the IDM Managed User repository to the MongoDB database (`managedUser_systemMongodbAccount`).
     3. Select the `managedUser_systemMongodbAccount` mapping, and choose the **Reconcile** option.
   - To reconcile the repository by using the command-line, launch the reconciliation operation with the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=managedUser_systemMongodbAccount&waitForCompletion=true"
{
  "_id": "e5bf074e-4da6-4ea7-8203-d4ec6f5a814a-24344",
  "state": "SUCCESS"
}
```

The reconciliation operation creates MongoDB users from the users found in `managed/user`.

Synchronize data between IDM and HubSpot

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

Prepare the sample

The sample assumes that you have a client app in HubSpot, with the corresponding `clientID`, `clientSecret`, and `refreshToken`, and that you already have some contacts stored in HubSpot.

Get the refresh token

1. Browse to the following URL:

   `https://app.hubspot.com/oauth/authorize?client_id=clientID&scope=contacts&redirect_uri=your-domain`

2. On the resulting page, select your user account in HubSpot. You are redirected to a URL similar to the following:

   `https://your-domain/?code=860c1867-9e4b-4761-82c4-1a5a4caf5224`

3. Copy the `code` in this URL (`860c1867-9e4b-4761-82c4-1a5a4caf5224` from the above example). You will need the code for the following step.

4. Send the following POST request, substituting your `clientID` and `clientSecret`:

```
curl \
--request POST \
--header 'Cache-Control: no-cache' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data 'grant_type=authorization_code&client_id=your-id&client_secret=your-secret&redirect_uri=your-
domain&code=code' \
 "https://api.hubapi.com/oauth/v1/token"
{
  "refresh_token": "f37e1132-xxxx-xxxx-xxxx-xxxx",
  "access_token": "CKbm...VaE",
  "expires_in": 21600
}
```

> **Note**
>
> The `redirect_uri` must be URL-encoded in this request, for example, `redirect_uri=https%3A%2F%2Fwww.example.com%2F`.

The output of the POST request includes the `refresh_token` required to configure the connector.

Run the sample

In this section, you will do the following:

1. Download and install the HubSpot connector.
2. Start IDM with the sample configuration.
3. Configure the HubSpot connector and test your connection to HubSpot.
4. Reconcile your HubSpot contacts with the IDM managed user repository.
5. Change a user in IDM and reconcile the changes back to HubSpot.

The mapping configuration file (`sync.json`) for this sample includes two mappings: `systemHubspotContact_managedUser` and `managedUser_systemHubspotContact`. You will use these mappings to reconcile users between IDM and HubSpot.

1. To install the HubSpot connector, download the connector jar from the Backstage download site site and place it in the `/path/to/openidm/connectors` directory:

   ```
   mv ~/Downloads/hubspot-connector-1.5.2.0.jar /path/to/openidm/connectors/
   ```

2. Start IDM with the configuration for the HubSpot sample:

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

3. To configure the HubSpot connector, do one of the following:

   - Update `samples/sync-with-hubspot/conf/provisioner.openicf-hubspot.json` with your HubSpot `clientID`, `clientSecret`, and `refreshToken`, and setting `"enabled" : true`.
   - Use the admin UI to configure the connector.

     1. Select **Configure > Connectors**, and select the HubSpot connector.
     2. Complete at least the **Base Connector Details** and enable the connector.

4. Test the connection to HubSpot by running the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=test"
[
  {
    "name": "hubspot",
    "enabled": true,
    "config": "config/provisioner.openicf/hubspot",
    "connectorRef": {
      "bundleVersion": "[1.5.0.0,1.6.0.0)",
      "bundleName": "org.forgerock.openicf.connectors.hubspot-connector",
      "connectorName": "org.forgerock.openicf.connectors.hubspot.HubspotConnector"
    },
    "displayName": "Hubspot Connector",
    "objectTypes": [
      "company",
      "contactProperties",
      "__ALL__",
      "companyProperties",
      "contact"
    ],
    "ok": true
  }
]
```

A status of `"ok": true` indicates that the connector can connect to HubSpot.

> **Note**
>
> If you configured the connector through the admin UI, the connection is tested as soon as you select Save.

5. To reconcile your HubSpot contacts with the IDM managed user repository, do one of the following:

   - Run the command:

     ```
     curl \
     --header "X-OpenIDM-Username: openidm-admin" \
     --header "X-OpenIDM-Password: openidm-admin" \
     --header "Accept-API-Version: resource=1.0" \
     --request POST \
     "http://localhost:8080/openidm/recon?
     _action=recon&mapping=systemHubspotContact_managedUser&waitForCompletion=true"
     {
       "_id": "1f148024-45b9-4dc1-9c3f-29976e02db00-3768",
       "state": "SUCCESS"
     }
     ```

   - In the admin UI, select **Configure > Mappings**, and then select **Reconcile** on the `systemHubspotContact_managedUser` mapping.

6. In the admin UI, select **Manage > User**, and verify that your HubSpot contacts have been created as IDM managed users.
7. In the admin UI, select **Manage > User**, select a user to edit, and change one of the user properties.
8. To reconcile the managed user repository with your HubSpot contacts, do one of the following:
    - Run the command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=managedUser_systemHubspotContact&waitForCompletion=true"
{
  "_id": "1f148024-45b9-4dc1-9c3f-29976e02db00-8700",
  "state": "SUCCESS"
}
```

    - In the admin UI, select **Configure > Mappings**, and then select **Reconcile** on the `managedUser_systemHubspotContact` mapping.
9. In HubSpot, verify that the contact was updated.

For more information about the HubSpot connector, refer to HubSpot connector.

Synchronize data between IDM and DocuSign

> **Important**
>
> This sample only works with DocuSign connector version 1.5.20.21 and lower. For more information, refer to the OpenICF documentation.

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

The sample assumes that you have downloaded and installed the DocuSign connector and its dependencies, as described in Install the DocuSign connector. The sample also assumes that you have the DocuSign account information required to configure the connector, as described in Before you start.

Run the sample

In this section, you will do the following:

1. Start IDM with the sample configuration.
2. Configure the DocuSign connector and test your connection to DocuSign.
3. Reconcile your DocuSign service accounts with the IDM managed user repository.
4. Change a user in IDM and reconcile the changes back to DocuSign.

The mapping configuration file (`sync.json`) for this sample includes two mappings: `systemDocusignAccount_managedUser` and `managedUser_systemDocusignAccount` . You will use these mappings to reconcile users between IDM and DocuSign.

1. Start IDM with the configuration for the DocuSign sample:

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

2. To configure the DocuSign connector, do one of the following:

   - Update `samples/sync-with-docusign/conf/provisioner.openicf-docusign.json` with your DocuSign account details.
   - Use the admin UI to configure the connector.

   Follow one of the procedures in Install the DocuSign connector to configure the connector.

3. Test the connection to DocuSign:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=test"
[
  {
    "name": "docusign",
    "enabled": true,
    "config": "config/provisioner.openicf/docusign",
    "connectorRef": {
      "bundleVersion": "1.5.0.0",
      "bundleName": "org.forgerock.openicf.connectors.docusign-connector",
      "connectorName": "org.forgerock.openicf.connectors.docusign.DocuSignConnector"
    },
    "displayName": "DocuSign Connector",
    "objectTypes": [
      "userSignature",
      "signingGroup",
      "__ALL__",
      "account",
      "contact"
    ],
    "ok": true
  }
]
```

   A status of `"ok": true` indicates that the connector can connect to DocuSign.

> **Note**
>
> If you configured the connector through the admin UI, the connection is tested as soon as you select **Save**.

4. To reconcile your existing DocuSign users with the IDM managed user repository, do one of the following:

   - Run the command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemDocusignAccount_managedUser&waitForCompletion=true"
{
  "_id": "7dac3ea9-c6be-4ff9-ae46-d8a0431949b3-7745",
  "state": "SUCCESS"
}
```

   - In the admin UI, select **Configure > Mappings**, and then select **Reconcile** on the `systemDocusignAccount_managedUser` mapping.

5. In the admin UI, select **Manage > User**, and verify that your DocuSign users have been created as IDM managed users.

6. In the admin UI, select **Manage > User**, select a user to edit, and change some user properties.

7. To reconcile the users in the managed user repository with your DocuSign users, do one of the following:

   - Run the command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=managedUser_systemDocusignAccount&waitForCompletion=true"
{
  "_id": "1f148024-45b9-4dc1-9c3f-29976e02db00-8700",
  "state": "SUCCESS"
}
```

   - In the admin UI, select **Configure > Mappings**, and then select **Reconcile** on the `managedUser_systemDocusignAccount` mapping.

8. In DocuSign, verify that the contact was updated.

For more information about the DocuSign connector, refer to DocuSign connector.

Synchronize data between IDM and a SCIM provider

This sample demonstrates bidirectional synchronization between IDM and accounts configured to the System for Cross-domain Identity Management. As noted on their website, "The System for Cross-domain Identity Management (SCIM) specification is designed to make managing user identities in cloud-based applications and services easier."

While this sample has been built to comply with SCIM 2.0 standards, it's been tested with a SCIM 1.1 provider.

This sample assumes you've configured SCIM on a third-party system. From that system you'll need the following configuration properties:

- OAuth 2.0 Client ID
- OAuth 2.0 Client Secret
- OAuth 2.0 Token
- SCIM Endpoint
- SCIM Version
- Properties that you want to reconcile from the SCIM provider

> **Note**
>
> Depending on your provider, you may want to modify the `sync.json` file for this sample to match the properties from the SCIM provider to appropriate properties for IDM.

For more information on the SCIM connector, including properties for the `provisioner.openicf-scim.json` file, refer to SCIM connector.

Run the sample

In this section, you will do the following:

- Start IDM with the sample configuration.
- Configure the SCIM connector and test your connection to the third-party SCIM provider.
- Reconcile your SCIM accounts with the IDM managed user repository.
- Change a user in IDM and reconcile the changes back to the third-party SCIM provider.
- Reconcile your SCIM roles with the IDM managed role repository.

The mapping configuration file (`sync.json`) for this sample includes four mappings, which you'll use to reconcile users and roles:

- `systemScimAccount_managedUser`
- `managedUser_systemScimAccount`
- `systemScimGroup_managedRole`
- `managedRole_systemScimGroup`

1. Start IDM with the configuration for the SCIM sample:

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

2. Configure the SCIM connector, in the following configuration file: `samples/sync-with-scim/conf/provisioner.openicf-scim.json` .

> **Note**
>
> Depending on the requirements of your third-party SCIM provider, it may be acceptable to have a `null` value for properties such as `user`, `password`, and `tokenEndpoint`.

3. Test the connection to your third-party SCIM provider with the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=test"
[
  {
    "name": "scim",
    "enabled": true,
    "config": "config/provisioner.openicf/scim",
    "connectorRef": {
      "bundleVersion": "1.5.20.14",
      "bundleName": "org.forgerock.openicf.connectors.scim-connector",
      "connectorName": "org.forgerock.openicf.connectors.scim.ScimConnector"
    },
    "displayName": "Scim Connector",
    "objectTypes": [
      "__ALL__",
      "account",
      "group"
    ],
    "ok": true
  }
]
```

A status of `"ok": true` indicates that the connector can connect to your third-party SCIM provider.

4. To reconcile your existing third-party SCIM users with the IDM managed user repository, do one of the following:

    - Run the command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemScimAccount_managedUser&waitForCompletion=true"
{
  "_id": "bdba3003-0c8a-4543-9efb-26269c78fa8b-96949",
  "state": "SUCCESS"
}
```

- In the admin UI, select **Configure > Mappings**, and select **Reconcile** on the `systemScimAccount_managedUser` mapping.

5. In the admin UI, select **Manage > User** and verify that the users from the third-party SCIM provider have been created as IDM managed users.
6. In the admin UI, select **Manage > User**, select a user to edit, and change one of the user properties.
7. To reconcile the users in the managed user repository with your SCIM users, do one of the following:
   - Run the command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=managedUser_systemScimAccount&waitForCompletion=true"
{
  "_id": "bdba3003-0c8a-4543-9efb-26269c78fa8b-104117",
  "state": "SUCCESS"
}
```

   - In the admin UI, select **Configure > Mappings**, and then select **Reconcile** on the `managedUser_systemScimAccount` mapping.
8. Verify that the contact was updated on your third-party SCIM provider.
9. Repeat the process with roles. To reconcile existing third-party SCIM roles with IDM managed roles, do one of the following:
   - Run the command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=systemScimGroup_managedRole&waitForCompletion=true"
{
  "_id": "7dac3ea9-c6be-4ff9-ae46-d8a0431949b3-7745",
  "state": "SUCCESS"
}
```

- In the admin UI, select **Configure > Mappings**, and select **Reconcile** on the `systemScimGroup_managedRole` mapping.

10. In the admin UI, select **Manage > Role**, select a role to edit, and add a user to that role.
11. To reconcile the roles in the managed user repository with your SCIM users, do one of the following:
    - Run the command::

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?
_action=recon&mapping=managedRole_systemScimGroup&waitForCompletion=true"
{
  "_id": "bdba3003-0c8a-4543-9efb-26269c78fa8b-112074",
  "state": "SUCCESS"
}
```

    - In the admin UI, select **Configure > Mappings**, and select **Reconcile** on the `managedRole_systemScimGroup` mapping.
12. Verify that the role was updated on your third-party SCIM provider.

Subscribe to JMS messages

IDM can subscribe to Java Messaging Service (JMS) messages using the Messaging Service's JMS Subscriber. In an event-driven architecture, also known as a message-driven architecture, there are publishers and subscribers. When a publisher sends a message over JMS, that message is broadcast. All active and subscribed clients receive that message. This sample shows how IDM can act as a JMS message subscriber, using the ActiveMQ Artemis JMS message broker.

> **Note**
>
> For more information on how IDM can publish JMS messages using the JMS Audit Event Handler, refer to Direct audit information to a JMS broker

Sample overview

With the scripted message handler shown in this sample, you can configure scripts to parse the contents of JMS messages, and act on that content.

The script in this sample, `crudpaqTextMessageHandler.js`, shows how JMS can handle ForgeRock REST operations. If you customize a script to manage JMS messages, you must also modify the `conf/messaging.json` file.

This sample uses ActiveMQ Artemis, a JMS message broker. With the ActiveMQ Artemis UI, you can act as the JMS message provider. This sample demonstrates how you can input REST payloads using the Artemis UI.

Dependencies for JMS messaging

The JMS audit event handler requires Apache ActiveMQ Artemis and additional dependencies bundled with the ActiveMQ Artemis delivery. This section lists the dependencies, and where they must be installed in the IDM instance. If you use a different ActiveMQ version, you may need to download the corresponding dependencies separately.

1. Download the following files:
   - Apache ActiveMQ Artemis.

     > **Note**
     >
     > This sample was tested with version 2.20.0.

   - The most recent `bnd` JAR file from https://repo1.maven.org/maven2/biz/aQute/bnd/biz.aQute.bnd/.

     > **Tip**
     >
     > The bnd utility lets you create OSGi bundles for libraries that do not support OSGi.

2. Unpack the ActiveMQ Artemis archive. For example:

```
tar -zxvf ~/Downloads/apache-artemis-2.20.0-bin.tar.gz
```

3. Create a temporary directory, and then change to that directory:

```
mkdir ~/Downloads/tmp
cd ~/Downloads/tmp/
```

4. Move the ActiveMQ Artemis Client and `bnd` JAR files to the temporary directory.

```
mv ~/Downloads/apache-artemis-2.20.0/lib/client/artemis-jms-client-all-2.20.0.jar ~/Downloads/tmp/
mv ~/Downloads/biz.aQute.bnd-version.jar ~/Downloads/tmp/
```

5. Create an OSGi bundle:
   1. In a text editor, create a BND file named `activemq.bnd` with the following contents, and save it to the current directory:

```
version=2.20.0
Export-Package: *;version=${version}
Import-Package: !org.apache.log4j.*,!org.apache.log.*,!org.apache.avalon.framework.logger.*,!
org.apache.avalon.framework.logger.*,!org.glassfish.json.*,!org.conscrypt.*,!
org.apache.logging.*,!org.bouncycastle.jsse.*,!org.eclipse.*,!sun.security.*,!reactor.*,!
org.apache.activemq.artemis.shaded.*,!com.aayushatharva.*,!com.github.luben.zstd,!
com.jcraft.jzlib,!com.ning.compress,!com.ning.compress.lzf,!com.ning.compress.lzf.util,!
com.oracle.svm.core.annotate,!lzma.*,!net.jpountz.*,*
Bundle-Name: ActiveMQArtemis :: Client
Bundle-SymbolicName: org.apache.activemq
Bundle-Version: ${version}
```

Your `tmp/` directory should now contain the following files:

```
ls -1 ~/Downloads/tmp/
activemq.bnd
artemis-jms-client-all-2.20.0.jar
biz.aQute.bnd-version.jar
```

2. In the same directory, create the OSGi bundle archive file. For example:

```
java -jar biz.aQute.bnd-version.jar wrap \
--properties activemq.bnd \
--output artemis-jms-client-all-2.20.0-osgi.jar \
artemis-jms-client-all-2.20.0.jar
```

6. Copy the resulting `artemis-jms-client-all-2.20.0-osgi.jar` file to the `openidm/bundle` directory:

```
cp artemis-jms-client-all-2.20.0-osgi.jar /path/to/openidm/bundle/
```

Configure SSL for Apache ActiveMQ Artemis

For information on configuring Apache ActiveMQ Artemis security features, including SSL, refer to the ActiveMQ Artemis Documentation:

- Security
- Configuring the Transport

Configure a secure port for JMS messages

If you configured SSL for ActiveMQ Artemis, edit `/path/to/openidm/samples/scripted-jms-subscriber/conf/messaging.json`, and replace the `java.naming.provider.url`:

```
"java.naming.provider.url" : "ssl://localhost:61617?daemon=true"
```

Start the ActiveMQ Artemis broker and IDM

With the appropriate bundles in the `/path/to/openidm/bundles` directory, you're ready to start the ActiveMQ Artemis message broker, as well as IDM with the JMS Audit Sample.

> **Important**
>
> For a full list of ActiveMQ Artemis setup options, refer to Using the Server in the *Artemis Documentation*.

1. Navigate to the directory where you unpacked the ActiveMQ Artemis binary and run the following command to create the Artemis broker:

```
cd ~/Downloads/apache-artemis-2.20.0/bin
./artemis create fr-scripted-jms
Creating ActiveMQ Artemis instance at: /path/to/Downloads/apache-artemis-2.20.0/bin/fr-scripted-jms
...
```

2. Start the newly created ActiveMQ Artemis broker:

```
./fr-scripted-jms/bin/artemis run
```

3. Start IDM, with the configuration for this sample:

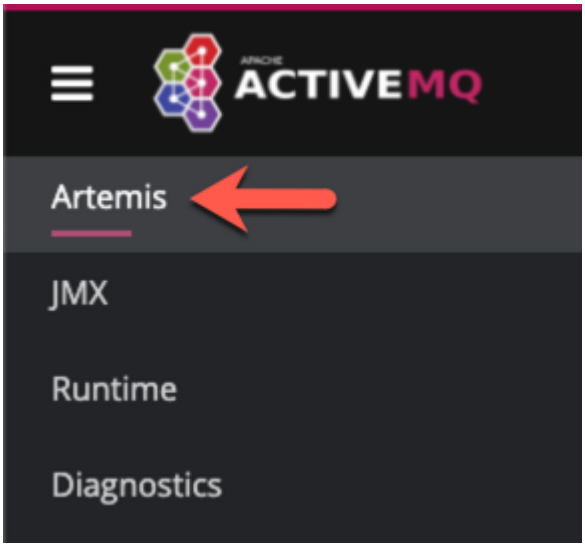
```
cd /path/to/openidm/
./startup.sh -p samples/scripted-jms-subscriber
```

4. Verify you can access the Artemis management console at `http://localhost:8161/console`.

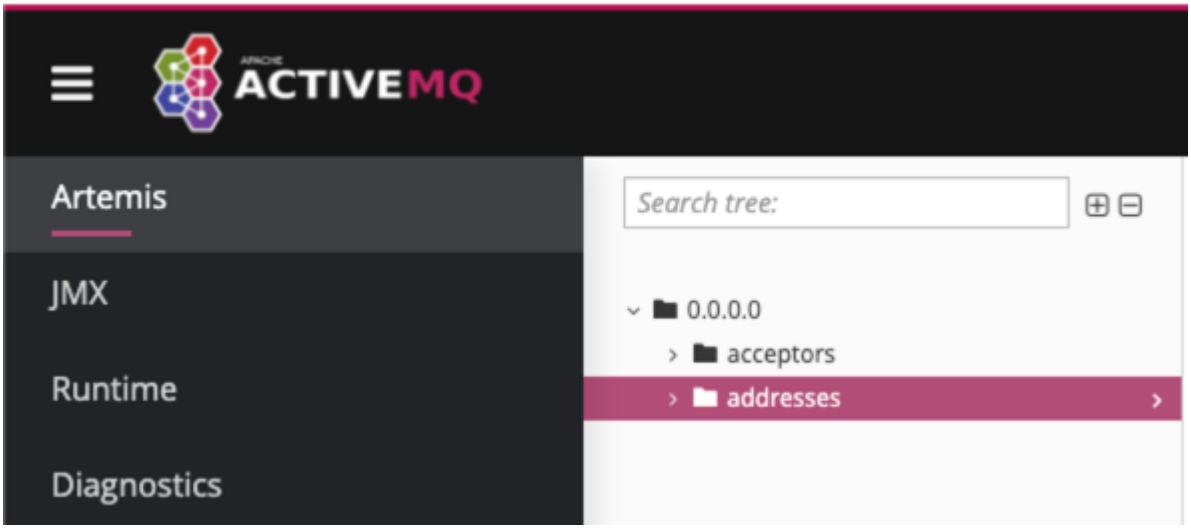
Use the ActiveMQ Artemis UI to access the REST interface

In this section, you will use the ActiveMQ Artemis UI to send REST requests.

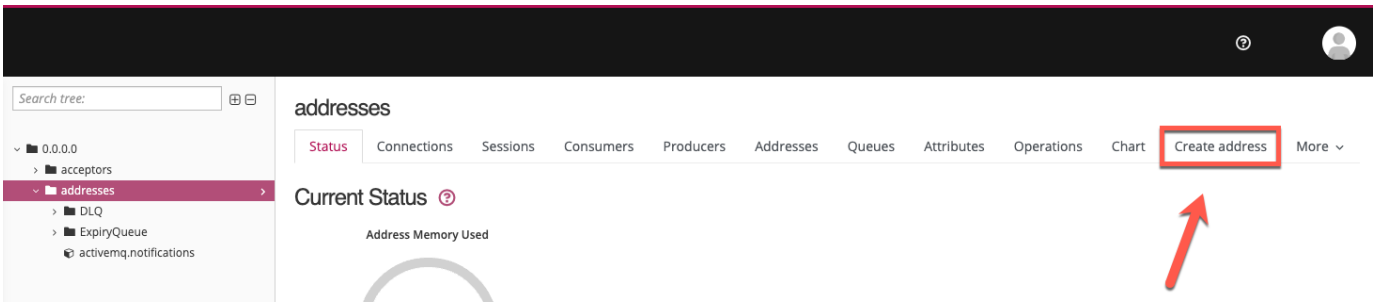
1. Log in to the Artemis management console (`http://localhost:8161/console`).
2. From the navigation menu, click **Artemis**.



3. From the tree view, select the **addresses** node.

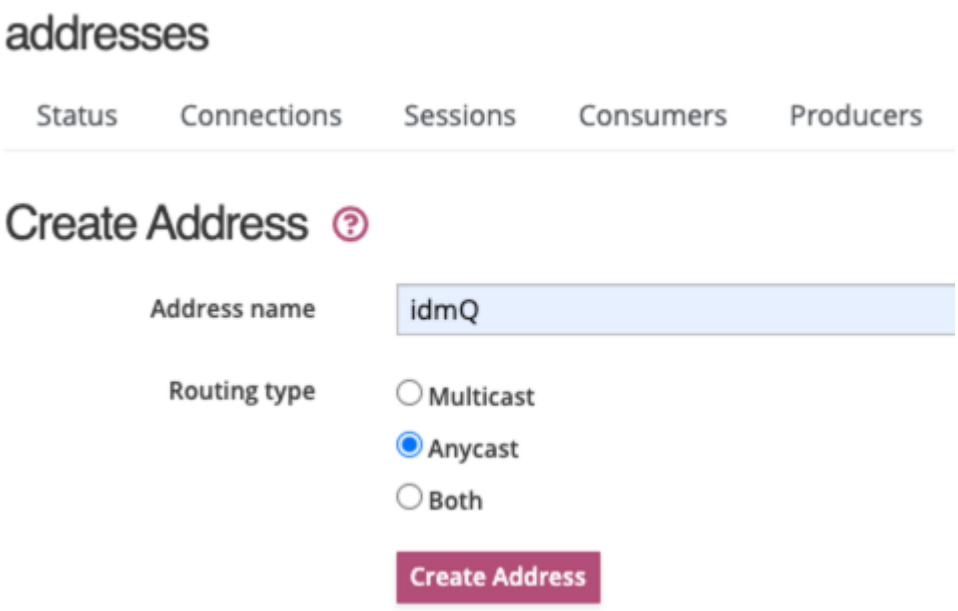


4. On the **addresses** page, click the **Create address** tab.



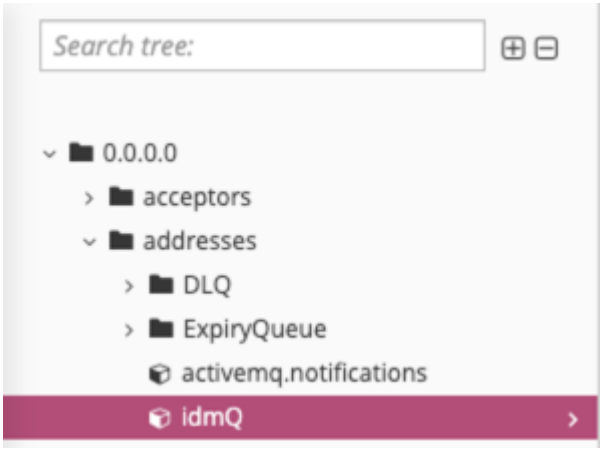
> **Note**
>
> Depending on your window size, the **Create address** tab may be located under the **More** menu.

5. Fill out the **Create Address** form, and then click **Create Address**:

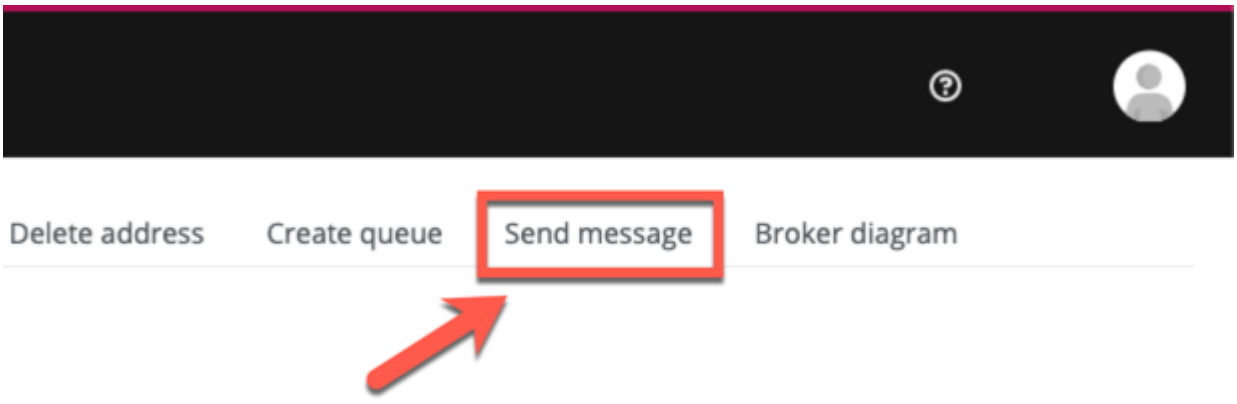


- **Address name**: `idmQ`
- **Routing type**: `Anycast`

6. From the tree view, expand the **addresses** node, and click **idmQ**.



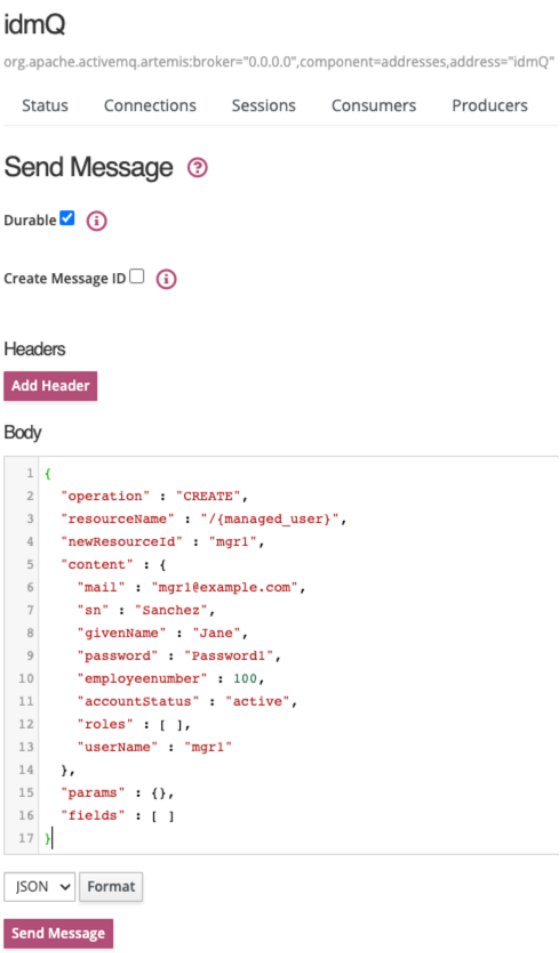
7. On the **idmQ** page, click the **Send message** tab.



> **Note**
>
> Depending on your window size, the **Send message** tab may be located under the **More** menu.

8. On the **Send Message** page, paste the following text into the **Body** field. This request creates a new user with a user ID of `mgr1`:

```
{
  "operation" : "CREATE",
  "resourceName" : "/managed/user",
  "newResourceId" : "mgr1",
  "content" : {
    "mail" : "mgr1@example.com",
    "sn" : "Sanchez",
    "givenName" : "Jane",
    "password" : "Password1",
    "employeenumber" : 100,
    "accountStatus" : "active",
    "roles" : [ ],
    "userName" : "mgr1"
  },
  "params" : {},
  "fields" : [ ]
}
```



For comparison, the following equivalent REST call would create the same user:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "mail" : "mgr1@example.com",
  "sn" : "Sanchez",
  "givenName" : "Jane",
  "password" : "Password1",
  "employeenumber" : 100,
  "accountStatus" : "active",
  "roles" : [ ],
  "userName" : "mgr1",
  "params" : {},
  "fields" : [ ]
}' \
"http://localhost:8080/openidm/managed/user?_action=create"
```

9. Click **Send Message**.

   The OSGi console displays the message request and response:

```
**************request received*************
Parsed JMS JSON =
{
    "operation": "CREATE",
    "resourceName": "/managed/user",
    "newResourceId": "mgr1",
    "content": {
        "mail": "mgr1@example.com",
        "sn": "Sanchez",
        "givenName": "Jane",
        "password": "Password1",
        "employeenumber": 100,
        "accountStatus": "active",
        "roles": [],
        "userName": "mgr1"
    },
    "params": {},
    "fields": []
}
Message response is...
{
    "accountStatus": "active",
    "password": {
        "$crypto": {
            "type": "x-simple-encryption",
            "value": { <encryptedValue> }
        }
    },
    "mail": "mgr1@example.com",
    "employeenumber": 100,
    "givenName": "Jane",
    "sn": "Sanchez",
    "userName": "mgr1",
    "effectiveRoles": [],
    "memberOfOrgIDs": [],
    "effectiveAssignments": [],
    "_rev": "17273eca-d14b-4647-b850-1e3733ba1830-116",
    "_id": "mgr1"
}
**************END MESSAGE*************
```

10. Confirm the user details:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/mgr1"
{
  "_id": "mgr1",
  "_rev": "17273eca-d14b-4647-b850-1e3733ba1830-116",
  "accountStatus": "active",
  "mail": "mgr1@example.com",
  "employeenumber": 100,
  "givenName": "Jane",
  "sn": "Sanchez",
  "userName": "mgr1",
  "effectiveRoles": [],
  "memberOfOrgIDs": [],
  "effectiveAssignments": []
}
```

11. You can repeat the process using different REST operations in the Artemis UI. For example, enter the following payload on the **Send Message** page to change the first name (`givenName`) of the `mgr1` user to `Donna` :

```
{
  "operation": "PATCH",
  "resourceName": "/managed/user/mgr1",
  "value": [
    {
      "operation": "replace",
      "field": "/givenName",
      "value": "Donna"
    }
  ]
}
```

12. Confirm the updated `givenName` for `mgr1` :

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/mgr1"
{
  "_id": "mgr1",
  "_rev": "17273eca-d14b-4647-b850-1e3733ba1830-315",
  "accountStatus": "active",
  "mail": "mgr1@example.com",
  "employeenumber": 100,
  "givenName": "Donna",
  "sn": "Sanchez",
  "userName": "mgr1",
  "effectiveRoles": [],
  "memberOfOrgIDs": [],
  "effectiveAssignments": []
}
```

Customize the scripted JMS sample

If you set up a custom script to parse and process JMS messages, store that script in the `script/` subdirectory. Assume the script is named `myCustomScript.js`.

Edit the `messaging.json` file in the `conf/` subdirectory, and point it to the custom file:

```
{
  "subscribers" : [
    {
      "name" : "IDM CREST Queue Subscriber",
      "instanceCount": 3,
      "enabled" : true,
      "type" : "JMS",
      "handler" : {
        "type" : "SCRIPTED",
        "properties" : {
          "script" : {
            "type" : "text/javascript",
            "file" : "myCustomScript.js"
          }
        }
      },
      "properties" : {
        "sessionMode" : "CLIENT",
        "jndi" : {
          "contextProperties" : {
            "java.naming.factory.initial" : "org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory",
            "java.naming.provider.url" : "tcp://127.0.0.1:61616?daemon=true",
            "queue.idmQ" : "idmQ"
          },
          "destinationName" : "idmQ",
          "connectionFactoryName" : "ConnectionFactory"
        }
      }
    }
  ]
}
```

You'll find some of these properties in JMS audit event handler properties. Despite the name of the table and the different configuration file, the properties are the same.

Other `messaging.json` notable properties:

JMS `messaging.json` *Configuration Properties*

`messaging.json` Property	Description
`subscribers`	Needed to subscribe to incoming JMS message requests.
`name`	Arbitrary name for the subscriber.
`instanceCount`	Each `instanceCount` manages a single connection between IDM and the messaging channel. Supports multithreading throughput. If subscribing to a queue, such as `queue.idmQ`, the message is handled by a single instance. If subscribing to a topic, all instances receive and handle the same message.
`handler`	Parses the JMS message, then processes it, possibly through a script.
`queue.idmQ`	One of the JNDI context properties. Name of the JMS queue in the Artemis UI.

`messaging.json` Property	Description
`destinationName`	JNDI lookup name for message delivery.

Authenticate using a trusted servlet filter

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

To configure authentication using ForgeRock Access Management, refer to the Platform Setup Guide.

Prepare the sample

Before you start this sample, complete the following steps:

1. Prepare a fresh IDM installation, as described in Prepare IDM.
2. Download and install the Apache Maven build tool.
3. Build the custom servlet filter bundle file:

```
cd /path/to/openidm/samples/trusted-servlet-filter/filter
mvn clean install
```

4. Copy the newly built servlet bundle file to the `openidm/bundle` directory:

```
cp target/sample-trusted-servletfilter-1.0.jar /path/to/openidm/bundle
```

The sample servlet filter

In the previous section, you built a bundle file from the Java file named `SampleTrustedServletFilter.java`, located in the directory `trustedservletfilter/filter/src/main/java/org/forgerock/openidm/sample/trustedservletfilter`.

The following line from the file looks for the `X-Special-Trusted-User` header, to identify a specific User ID as a "trusted" user.

```
final String specialHeader = ((HttpServletRequest) servletRequest).getHeader("X-Special-Trusted-User");
```

The next line sets the special Servlet attribute `X-ForgeRock-AuthenticationId` to this trusted User ID.

```
servletRequest.setAttribute("X-ForgeRock-AuthenticationId", specialHeader);
```

The rest of the servlet filter chain continues request processing:

```
filterChain.doFilter(servletRequest, servletResponse);
```

This sample includes a `servletfilter-trust.json` file that calls the compiled IDM trusted servlet `filterClass`:

```
{
   "classPathURLs" : [ ],
   "systemProperties" : { },
   "requestAttributes" : { },
   "scriptExtensions" : { },
   "initParams" : { },
   "urlPatterns" : [
      "/*"
   ],
   "filterClass" : "org.forgerock.openidm.sample.trustedservletfilter.SampleTrustedServletFilter"
}
```

Run the sample

In this section, you will demonstrate the servlet filter by configuring it with the special header described in the previous section. Normally, a servlet filter used for authentication does not let a client masquerade as any user. This sample demonstrates a basic use of a servlet filter by establishing the authentication ID.

1. Start IDM with the configuration for the trusted filter sample:

   ```
   cd /path/to/openidm/
   ./startup.sh -p samples/trusted-servlet-filter
   ```

2. Create a new managed user, Barbara Jensen, with userName `bjensen`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '
   {
      "userName": "bjensen",
      "telephoneNumber": "6669876987",
      "givenName": "Barbara",
      "sn": "Jensen",
      "description": "Example User",
      "mail": "bjensen@example.com",
      "authzRoles" : [
         {
            "_ref" : "internal/role/openidm-authorized"
         }
      ]
   }' \
"http://localhost:8080/openidm/managed/user"
{
  "_id": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
  "_rev": "0000000049889176",
  "userName": "bjensen",
  "telephoneNumber": "6669876987",
  "givenName": "Barbara",
  "sn": "Jensen",
  "description": "Example User",
  "mail": "bjensen@example.com",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

Note the ID of the new user—you will need it in the steps that follow.

3. Use the special request header `X-Special-Trusted-User` to authenticate bjensen (specifying her ID as the header value).

```
curl \
--header "X-Special-Trusted-User: 9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/info/login?_fields=authenticationId,authorization"
{
  "_id": "login",
  "authenticationId": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
  "authorization": {
    "component": "managed/user",
    "roles": [
      "internal/role/openidm-authorized"
    ],
    "id": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
    "moduleId": "TRUSTED_ATTRIBUTE"
  }
}
```

Note that the output includes the user's authentication and authorization details. In this case, bjensen, with ID `9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb` is authenticated with the `openidm-authorized` role.

Customize the sample for an external system

To customize this sample for an external authentication/authorization system, you need a servlet filter which authenticates against that external system. You may use a third-party supplied filter, or develop your own filter, using the one in this sample as a model.

The filter you use should have at least the following capabilities:

- Perform REST calls to another system.
- Search through databases.
- Inspect headers related to authentication and authorization requests.

This servlet filter must set the username of the authenticated user in a special request attribute. You need to configure that same attribute name in the `TRUSTED_ATTRIBUTE` authentication module, specifically the value of `authenticationIdAttribute`.

It is helpful if you have a filter that returns an object with the `userRoles` property. If your filter does not support queries using the following parameter:

```
queryOnResource + "/" + authenticationId
```

You will need to provide a security context augmentation script that populates the following authorization properties in the "security" object:

- `security.authorization.component`
- `security.authorization.roles`

The value for the `security.authorization.component` is automatically set to the value specified in any existing `queryOnResource` property.

Create a custom endpoint

Scriptable custom endpoints let you launch arbitrary scripts using the IDM REST URI. For information about how custom endpoints are configured, refer to Create custom endpoints to launch scripts.

The example endpoint provided in `/path/to/openidm/samples/example-configurations/custom-endpoint` illustrates the configuration of a custom endpoint, and the structure of custom endpoint scripts.

The purpose of this custom endpoint is to return a list of variables available to each method used in a script. The scripts show the complete set of methods that can be used. These methods map to the standard HTTP verbs - create, read, update, delete, patch, query, and action. A sample JavaScript and Groovy script are provided.

Run the sample

1. Copy the endpoint configuration file (`samples/example-configurations/custom-endpoint/conf/endpoint-echo.json`) to your project's `conf` directory.
2. Copy either the JavaScript file (`samples/example-configurations/custom-endpoint/script/echo.js`) or Groovy script file (`samples/example-configurations/custom-endpoint/script/echo.groovy`) to your project's `script` directory.
3. Open the endpoint configuration file in a text editor:

   ```
   {
       "file" : "echo.groovy",
       "type" : "groovy",
       "_file" : "echo.js",
       "_type" : "text/javascript",
       ...
   }
   ```

   The configuration file contains a reference to the endpoint scripts. In this case, the JavaScript script is commented out (with an underscore before the `file` and `type` properties). If you want to use the JavaScript endpoint script, uncomment these lines and comment out the lines that correspond to the Groovy script in the same way.

   Endpoint configuration files can include a `context` property that specifies the route to the endpoint, for example:

   ```
   "context" : "endpoint/linkedView/*"
   ```

   If no `context` is specified, the route to the endpoint is taken from the file name, in this case `endpoint/echo` .
4. Test each method in succession to return the expected request structure of that method. The following examples show the request structure of the read, create and patch methods. The configuration file has been edited to use the JavaScript file, rather than the Groovy file. The output shown in these examples has been cropped for legibility. For a description of each parameter, refer to Custom Endpoint Scripts.
   - The following command performs a read on the echo endpoint and returns the request structure of a read request:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/endpoint/echo"
{
  "_id": "",
  "method": "read",
  "context": {
    "class": "org.forgerock.http.routing.ApiVersionRouterContext",
    "name": "apiVersionRouter",
    "defaultVersionBehaviour": "LATEST",
    "warningEnabled": false,
    "resourceVersion": "1.0",
    "parent": {
      "class": "org.forgerock.http.routing.UriRouterContext",
      "name": "router",
      ...
    }
  },
  "resourceName": "",
  "parameters": {}
}
```

- The following command performs a query on the echo endpoint and returns the request structure of a query request:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/endpoint/echo?_queryFilter=true"
{
  "result": [
    {
      "method": "query",
      "pageSize": 0,
      "queryFilter": "true",
      "resourceName": "",
      "pagedResultsOffset": 0,
      "pagedResultsCookie": null,
      "parameters": {},
      "content": null,
      "queryId": null
      "content": null,
      "context": {
      ...
      }
    }
  ],
  ...
}
```

- The following command sends a create request to the echo endpoint. No user is actually created. The endpoint script merely returns the request structure of a create request. The `content` parameter in this case provides the JSON object that was sent with the request:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--data '{
       "userName":"steve",
       "givenName":"Steve",
       "sn":"Carter",
       "telephoneNumber":"0828290289",
       "mail":"scarter@example.com",
       "password":"Passw0rd"
       }' \
--request POST \
"http://localhost:8080/openidm/endpoint/echo?_action=create"
{
  "_id": "",
  "method": "create",
  "resourceName": "",
  "newResourceId": null,
  "parameters": {},
  "content": {
    "userName": "steve",
    "givenName": "Steve",
    "sn": "Carter",
    "telephoneNumber": "0828290289",
    "mail": "scarter@example.com",
    "password": "Passw0rd"
  },
  "context": {
  ...
  }
}
```

- The following command sends a patch request to the echo endpoint.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--data '[
    {
      "operation":"replace",
      "field":"/givenName",
      "value":"Steven"
    }
 ]' \
--request PATCH \
"http://localhost:8080/openidm/endpoint/echo"
{
  "_id": "",
  "method": "patch",
  "resourceName": "",
  "revision": null,
  "patch": [
    {
      "operation": "replace",
      "field": "/givenName",
      "value": "Steven"
    }
  ],
  "parameters": {},
  "context": {
  …
  }
}
```

Installation

> *Guide to installing, and uninstalling ForgeRock® Identity Management software. This software offers flexible services for automating management of the identity life cycle.*

This guide shows you how to install ForgeRock Identity Management services for identity management, provisioning, and compliance. You do not need a complete understanding of ForgeRock Identity Management software to learn something from this guide, although a background in identity management and maintaining web application software can help. You do need some background in managing services on your operating systems and in your web application containers. Unless you are planning an evaluation or test installation, read the Release notes before you get started.

Quick Start

Install IDM

Download the IDM software and get a minimal deployment up and running.

Interact with IDM

Introduction to the IDM REST API and browser-based user interface.

Repository

Configure IDM to use your selected production repository.

Startup Configuration

Learn about the startup configuration and how to verify system health.

Install in a Cluster

IDM in a cluster for availability.

Jetty Configuration

Configure the embedded Jetty server.

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

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

Java requirements

Important

In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
Learn more in What's new.

Before you start, follow these steps to ensure that your Java environment is suitable:

1. Verify that your computer has a supported Java version installed:

Supported Java Versions

Vendor	Versions
OpenJDK, including OpenJDK-based distributions: ◦ AdoptOpenJDK/Eclipse Temurin ◦ Amazon Corretto ◦ Azul Zulu ◦ Red Hat OpenJDK **Note** ForgeRock tests most extensively with AdoptOpenJDK/Eclipse Temurin. ForgeRock recommends using the HotSpot JVM.	17*

Vendor	Versions
Oracle Java	17*

* Version 17.0.9 or higher.

2. Read the pre-installation requirements.
3. Set the `JAVA_HOME` environment variable:
    1. Locate the JRE installation directory (typically, `C:\Program Files\Java\`).
    2. Click **Start > Control Panel > System and Security > System**.
    3. Click **Advanced System Settings**.
    4. Click **Environment Variables**.
    5. Under **System Variables**, click **New**.
    6. Enter the Variable name (`JAVA_HOME`) and set the Variable value to the JRE installation directory; for example `C:\Program Files\Java\jre8`.
    7. Click **OK**.
    1. Open the user shell configuration file found in your home directory.
    2. Add the `JAVA_HOME` variable to the user shell configuration file, setting the value to `/usr`. In Bash, this would appear as `export JAVA_HOME="/usr"`.

Install and run IDM

Use the procedures in this section to install, start, run, and stop IDM.

Follow these steps to install IDM:

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

```
java -version
openjdk version "11.0.6" 2020-01-14
OpenJDK Runtime Environment AdoptOpenJDK (build 11.0.6+10)
OpenJDK 64-Bit Server VM AdoptOpenJDK (build 11.0.6+10, mixed mode)
```

   For a description of the Java requirements, refer to Before you install.

2. Download IDM from Backstage. Releases on Backstage are thoroughly validated for ForgeRock customers who run the software in production deployments, and for those who want to try or test a given release.
3. Unpack the contents of the .zip file into the install directory:

```
unzip ~/Downloads/IDM-7.3.2.zip
Archive:  IDM-7.3.2.zip
  inflating: openidm/.checksums.csv
   creating: openidm/bundle/
 extracting: openidm/bundle/openidm-audit-7.3.2.jar
...
```

4. By default, IDM listens for HTTP and HTTPS connections on ports 8080 and 8443, respectively. To change these port numbers, edit the following settings in your `resolver/boot.properties` file:

   - `openidm.port.http`
   - `openidm.port.https`

   When you deploy IDM in production, you *must* set `openidm.host` to the URL of your deployment, in the same `resolver/boot.properties` file. Otherwise, calls to the `/admin` endpoint are not properly redirected.

   Deployment URLs will vary, depending on whether you're using a load balancer. While IDM documentation does not specify how you'd configure a load balancer, you'll need to configure IDM in a cluster as described in IDM cluster configuration, and specifically in Deploy Securely Behind a Load Balancer.

5. Before running IDM in production, replace the default embedded DS repository with a supported repository.

   For more information, refer to Select a repository.

Follow these steps to run IDM interactively:

1. Start the Felix container, load all services, and start a command shell to allow you to manage the container:

Bash

```
/path/to/openidm/startup.sh
Using OPENIDM_HOME:   /path/to/openidm
Using PROJECT_HOME:   /path/to/openidm
Using OPENIDM_OPTS:   -Xmx1024m -Xms1024m
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
-> OpenIDM version "7.3.2"
OpenIDM ready
```

PowerShell

```
\path\to\openidm\startup.bat
"Using OPENIDM_HOME:   \path\to\openidm"
"Using PROJECT_HOME:   \path\to\openidm"
"Using OPENIDM_OPTS:   -Xmx1024m -Xms1024m -Dfile.encoding=UTF-8"
"Using LOGGING_CONFIG: -
Djava.util.logging.config.file=\path\to\openidm\conf\logging.properties"
-> OpenIDM version "7.3.2"
OpenIDM ready
```

2. At the OSGi console `→` prompt, you can enter commands such as `help` for usage, or `ps` to view the bundles installed.

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

You can also start IDM as a background process on UNIX and Linux systems. Follow these steps, preferably before you start IDM for the first time:

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

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

2. Start the server in the background. The `nohup` survives a logout, and the `2>&1&` redirects standard output and standard error to the noted `console.out` file:

   ```
   nohup ./startup.sh > logs/console.out 2>&1&
   [1] 2343
   ```

To stop the server running as a background process, use the `shutdown.sh` script:

```
./shutdown.sh
Stopping OpenIDM (2343)
```

> **Note**
>
> Although installations on macOS systems are not supported in production, you might want to run IDM on macOS in a demo or test environment. To run IDM in the background on a macOS system, take the following additional steps:
>
> - Remove the `org.apache.felix.shell.tui-*.jar` bundle from the `openidm/bundle` directory.
> - Disable ConsoleHandler logging.

You can stop IDM from the `->` prompt in the OSGi console.

- In the OSGi console, enter the `shutdown` command at the `->` prompt.

- On Unix systems, you can stop IDM by using the `shutdown.sh` script:

```
/path/to/openidm/shutdown.sh
Stopping OpenIDM (31391)
```

1. Stop the server if it is running, as described in Stop IDM.
2. Remove the directory where you installed the software:

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

3. If you use a JDBC database for the repository, drop the `openidm` database.

To debug custom libraries, start the server with the Java Platform Debugger Architecture (JPDA):

1. Start IDM with the `jpda` option:

```
/path/to/openidm/startup.sh jpda
Executing ./startup.sh...
Using OPENIDM_HOME:   /path/to/openidm
Using OPENIDM_OPTS:   -Xmx1024m -Xms1024m -Djava.compiler=NONE -Xnoagent -Xdebug
 -Xrunjdwp:transport=dt_socket,address=5005,server=y,suspend=n
Using LOGGING_CONFIG:
   -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
Listening for transport dt_socket at address: 5005
Using boot properties at /path/to/openidm/resolver/boot.properties
-> OpenIDM version "7.3.2" (revision: xxxx)
OpenIDM ready
```

   The relevant JPDA options are listed in the startup script (`startup.sh`).

2. In your IDE, attach a Java debugger to the JVM via socket on port 5005.

> **Caution**
>
> This interface is internal and subject to change. If you depend on this interface, contact support.

Interact with IDM

There are two primary ways to interact with IDM; programmatically, using REST to access IDM's API endpoints, or using the browser-based user interfaces.

REST interface introduction

IDM provides RESTful access to users in its repository, and to its configuration. To access the repository over REST, you can use a browser-based REST client, such as the *Simple REST Client* for Chrome, or *RESTClient* for Firefox. You can also use applications such as *Postman* to create, run, and manage collections of REST calls. Alternatively you can use the `curl` command-line utility, included with most operating systems. For more information about `curl`, refer to https://github.com/curl/curl.

IDM is accessible over the regular and secure HTTP ports of the Jetty Servlet container, 8080, and 8443. Most of the command-line examples in this documentation set use the regular HTTP port, so that you don't have to use certificates just to test IDM. In a production deployment, install a CA-signed certificate and restrict REST access to a secure (HTTPS) port.

To run `curl` over the secure port, 8443, you must either include the `--insecure` option, or follow the instructions in Restrict REST Access to the HTTPS Port. You can use those instructions with the self-signed certificate that is generated when IDM starts, or with a `*.crt` file provided by a certificate authority.

> **Note**
>
> Some of the examples in this documentation set use client-assigned IDs (such as `bjensen` and `scarter`) when creating objects because it makes the examples easier to read. If you create objects using the admin UI, they are created with server-assigned IDs (such as `55ef0a75-f261-47e9-a72b-f5c61c32d339`). Generally, immutable server-assigned UUIDs are used in production environments.

1. Use the following REST query to list all users in the IDM repository:

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

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

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

   The following `curl` command creates a managed user in the repository, and set the user's ID to `jdoe`:

Bash

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "userName": "joe",
  "givenName": "joe",
  "sn": "smith",
  "mail": "joe@example.com",
  "telephoneNumber": "555-123-1234",
  "password": "TestPassw0rd",
  "description": "My first user",
  "_id": "joe"
}' \
http://localhost:8080/openidm/managed/user?_action=create
{
  "_id": "joe",
  "_rev": "00000000c03fd7aa",
  "userName": "joe",
  "givenName": "joe",
  "sn": "smith",
  "mail": "joe@example.com",
  "telephoneNumber": "555-123-1234",
  "description": "My first user",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

PowerShell

```
curl `
--header "Content-Type: application/json" `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0" `
--request POST `
--data '{
  "userName": "joe",
  "givenName": "joe",
  "sn": "smith",
  "mail": "joe@example.com",
  "telephoneNumber": "555-123-1234",
  "password": "TestPassw0rd",
  "description": "My first user",
  "_id": "joe"
}' `
http://localhost:8080/openidm/managed/user?_action=create
{
  "_id": "joe",
  "_rev": "00000000c03fd7aa",
  "userName": "joe",
  "givenName": "joe",
  "sn": "smith",
  "mail": "joe@example.com",
  "telephoneNumber": "555-123-1234",
  "description": "My first user",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

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

Bash

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
http://localhost:8080/openidm/managed/user/joe
{
  "_id": "joe",
  "_rev": "00000000c03fd7aa",
  "userName": "joe",
  "givenName": "joe",
  "sn": "smith",
  "mail": "joe@example.com",
  "telephoneNumber": "555-123-1234",
  "description": "My first user",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

PowerShell

```
curl `
--header "X-OpenIDM-Username: openidm-admin" `
--header "X-OpenIDM-Password: openidm-admin" `
--header "Accept-API-Version: resource=1.0" `
--request GET `
http://localhost:8080/openidm/managed/user/joe
{
  "_id": "joe",
  "_rev": "00000000c03fd7aa",
  "userName": "joe",
  "givenName": "joe",
  "sn": "smith",
  "mail": "joe@example.com",
  "telephoneNumber": "555-123-1234",
  "description": "My first user",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

Format REST output for readability

By default, `curl`-based REST calls return the JSON object on one line, which can be difficult to read. For example:

```
{"mail":"joe@example.com","sn":"smith","passwordAttempts":"0",
"lastPasswordAttempt":"Mon Apr 14 2014 11:13:37 GMT-0800 (GMT-08:00)",
"givenName":"joe","effectiveRoles":["internal/role/openidm-authorized"],
"password":{"$crypto":{"type":"x-simple-encryption","value":{"data":
"OBFVL9cG8uaLoo1N+SMJ3g==","cipher":"AES/CBC/PKCS5Padding","iv":
"7rlV4EwkwdRHkt19F8g22A==","key":"openidm-sym-default"}}},"country":"",
"city":"","_rev": "00000000c03fd7aa","lastPasswordSet":"","postalCode":"",
"_id":"joe3","description":"My first user","accountStatus":"active","telephoneNumber":
"555-123-1234","roles":["internal/role/openidm-authorized"],"effectiveAssignments":{},
"postalAddress":"","stateProvince":"","userName":"joe3"}
```


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

The standard way to format JSON output is with a JSON parser such as jq. `jq` is not installed by default on most operating systems, but you can install it and then "pipe" the output of a REST call to `jq`, as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/joe" \
| jq .
```

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

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

> **Note**
>
> Most command-line examples in this guide do not show this parameter, but the output is formatted for readability.

IDM user interfaces

IDM provides UIs at two different endpoints; `/` and `/admin`. The administrative tools available at `/admin` are called the admin UI. The End User UI enables end users to manage certain aspects of their own accounts.

For information about the admin UI, refer to IDM user interface .

For information about the End User UI, refer to Self-service end user UI.

IDM as a service

IDM as a Linux service

IDM provides a script that can generate `SysV` or `Systemd` service initialization scripts. You can start the script as the root user, or configure it to start during the boot process.

When IDM runs as a service, logs are written to the installation directory.

1. If you have not yet installed IDM, follow the steps in Install IDM.
2. Review the options by running the following script:

```
/path/to/openidm/bin/create-openidm-rc.sh
Usage: ./create-openidm-rc.sh --[systemd|chkconfig|lsb]
Outputs OpenIDM init file to stdout for the given system

--systemd    Generate Systemd init script. This is preferred for all modern distros.
--chkconfig  Generate SysV init script with chkconfig headers (RedHat/CentOS)
--lsb        Generate SysV init script with LSB headers (Debian/Ubuntu)
...
```

These examples describe how to create each of these scripts:

If you're running relatively standard versions of Red Hat Enterprise Linux (CentOS Linux) version 7.x, or Ubuntu 16.04 and later, you'll want to set up a systemd service script. To set up such a script, navigate to the `/path/to/openidm/bin` directory, and run the following command:

```
/path/to/openidm/bin/create-openidm-rc.sh --systemd
```

As noted in the output, you can set up the IDM service on a standard systemd-based Linux distribution with the following commands:

```
/path/to/openidm/bin/create-openidm-rc.sh --systemd > openidm.service
sudo cp openidm.service /etc/systemd/system/
systemctl enable openidm
systemctl start openidm
```

To stop the IDM service, run the following command:

```
systemctl stop openidm
```

You can modify the `openidm.service` script. The following excerpt would run IDM with a startup script in the `/home/idm/project` directory:

```
[Unit]
Description=ForgeRock OpenIDM
After=network.target auditd.target

[Service]
Type=simple
SuccessExitStatus=143
Environment=JAVA_HOME=/usr
User=testuser
ExecStart=/root/openidm/startup.sh -p /home/idm/project
ExecStop=/root/openidm/shutdown.sh

[Install]
WantedBy=multi-user.target
```

Run the following command to reload the configuration and then start the IDM service script:

```
systemctl daemon-reload
systemctl start openidm
```

If you are running standard versions of Red Hat Enterprise Linux (CentOS Linux) version 6.x, set up a SysV service script with runlevels controlled through the `chkconfig` command. To set up such a script, run the following command:

```
/path/to/openidm/bin/create-openidm-rc.sh --chkconfig
```

You can then set up and start the IDM service on a Linux distribution that uses SysV init scripts, with the following commands:

```
/path/to/openidm/bin/create-openidm-rc.sh --chkconfig  > openidm
sudo cp openidm /etc/init.d/
sudo chmod u+x /etc/init.d/openidm
sudo chkconfig --add openidm
sudo chkconfig openidm on
sudo service openidm start
```

To stop the IDM service, run the following command:

```
sudo service openidm stop
```

You can modify the `/etc/init.d/openidm` script. The following excerpt would run IDM with the `startup.sh` script in the `/path/to/openidm` directory:

```
START_CMD="PATH=$JAVA_BIN_PATH:$PATH;nohup $OPENIDM_HOME/startup.sh >$OPENIDM_HOME/logs/server.out 2>&1 &"
```

You can modify this line to point to some `/path/to/production` directory:

```
START_CMD="PATH=$JAVA_BIN_PATH:$PATH;nohup $OPENIDM_HOME/startup.sh -p /path/to/production >$OPENIDM_HOME/
logs/server.out 2>&1 &"
```

Run the following command to reload the configuration and then start the IDM service script:

```
sudo service openidm start
```

If you run Linux with SELinux enabled, change the file context of the newly copied script with the following command:

```
sudo restorecon /etc/init.d/openidm
```

Verify the change to SELinux contexts with the `ls -Z /etc/init.d` command. For consistency, change the user context to match other scripts in the same directory with the `sudo chcon -u system_u /etc/init.d/openidm` command.

If you're running an older version of Ubuntu Linux that supports SysV services, set up a SysV service script, with runlevels controlled through the `update-rc.d` command. To set up such a script, run the following command:

```
/path/to/openidm/bin/create-openidm-rc.sh --lsb
```

You can then set up and start the IDM service on a Linux distribution that uses SysV init scripts, with the following commands:

```
/path/to/openidm/bin/create-openidm-rc.sh --lsb  > openidm
sudo cp openidm /etc/init.d/
sudo chmod u+x /etc/init.d/openidm
sudo update-rc.d openidm defaults
sudo service openidm start
```

To stop the IDM service, run the following command:

```
sudo service openidm stop
```

You can modify the `/etc/init.d/openidm` script. The following excerpt would run IDM with the `startup.sh` script in the `/path/to/openidm` directory:

```
START_CMD="PATH=$JAVA_BIN_PATH:$PATH;nohup $OPENIDM_HOME/startup.sh >$OPENIDM_HOME/logs/server.out 2>&1 &"
```

You can modify this line to point to some `/path/to/production` directory:

```
START_CMD="PATH=$JAVA_BIN_PATH:$PATH;nohup $OPENIDM_HOME/startup.sh -p /path/to/production >$OPENIDM_HOME/
logs/server.out 2>&1 &"
```

You can then run the following command to reload the configuration and then start the IDM service script:

```
sudo service openidm restart
```

IDM as a Windows service

You can install IDM to run as a Windows service so that it automatically starts and stops with Windows. You must be logged in as an administrator to install a Windows service.

> **Note**
>
> On a 64-bit Windows server, you must have a 64-bit Java version installed to start the service. If a 32-bit Java version is installed, you will be able to install IDM as a service, but starting the service will fail.
> *Before* you launch the `service.bat` file, which registers the service within the Windows registry, make sure that your `JAVA_HOME` environment variable points to a valid 64-bit version of the JRE or JDK. If you have already installed the service with the `JAVA_HOME` environment variable pointing to a 32-bit JRE or JDK, delete the service first, then reinstall the service.

1. Unpack the IDM-7.3.2.zip file, as described previously, and navigate to the `install-directory\bin` directory:

```
C:\> cd openidm\bin
C:\openidm\bin>
```

2. Run the `service.bat` command with the `/install` option, specifying the name that the service should run as:

```
C:\openidm\bin> service.bat /install openidm
ForgeRock Identity Management Server successfully installed as "openidm" service
```

3. Use the Windows Service manager to manage the IDM service.

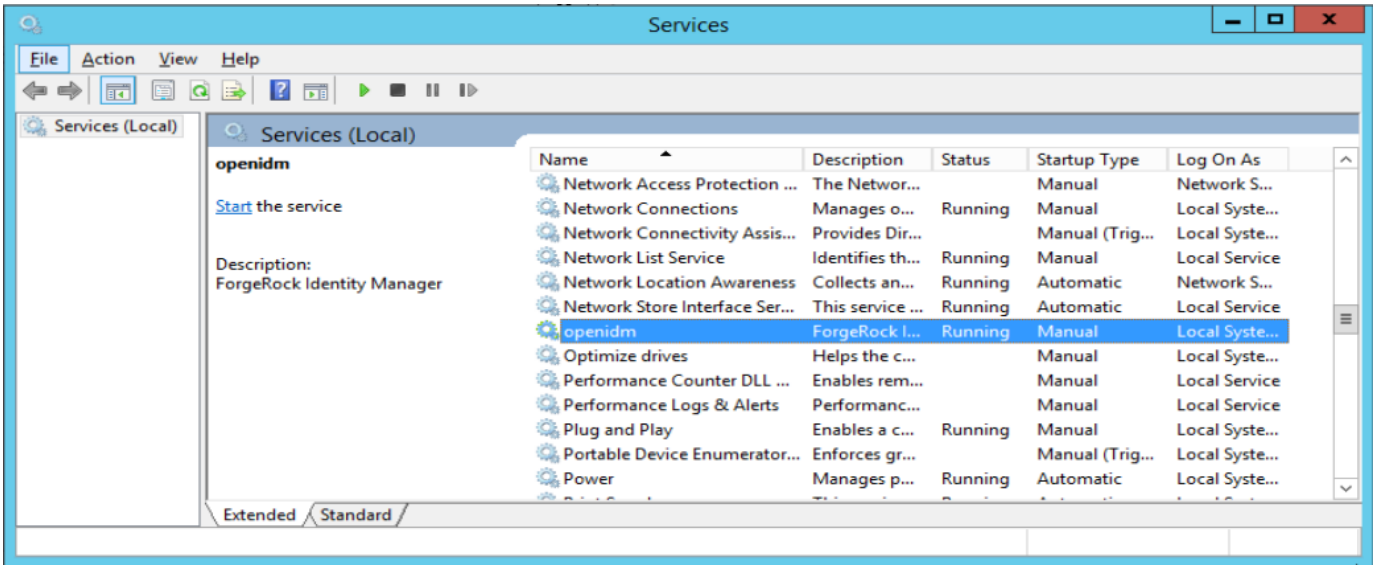


Figure 1. Running as a Windows Service

4. By default, the IDM service is run by `Local System`, which is a system-level service account built in to Windows. Before you deploy IDM in production, you should switch to an account with fewer permissions. The account running the IDM service must be able to read, write, and execute only the directories related to IDM.
5. Use the Windows Service Manager to start, stop, or restart the service.
6. If you want to uninstall the IDM service, first use the Windows Service Manager to stop IDM and then run the following command:

```
C:\install-directory\openidm\bin> service.bat /uninstall openidm
Service "openidm" removed successfully
```

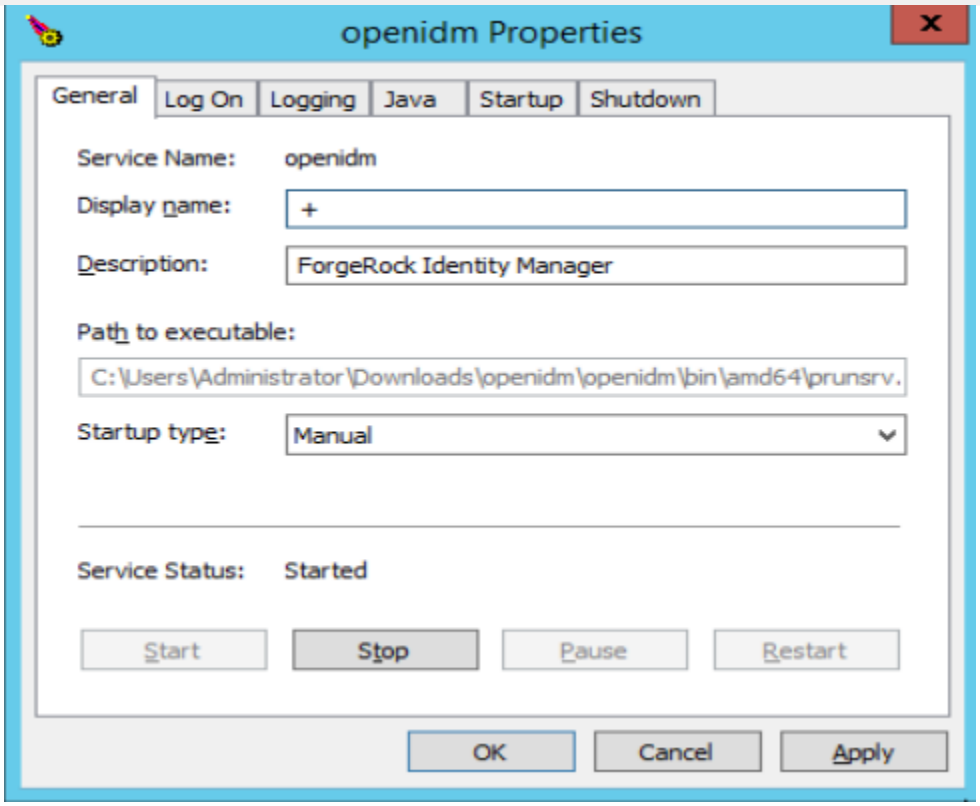
7. If desired, you can then set up IDM with a specific project directory:

```
C:\install-directory\openidm\bin> service.bat /install openidm -p C:\project-directory
ForgeRock Identity Management Server successfully installed as "openidm" service
```

You can also manage configuration details with the Procrun monitor application. IDM includes the associated `prunmgr.exe` executable in the `C:\install-directory\openidm\bin` directory.

For example, you can open the Windows service configuration application for IDM with the following command, where `ES` stands for *Edit Service Configuration*

```
C:\install-directory\openidm\bin> prunmgr.exe //ES/openidm
```



The `prunmgr.exe` executable also includes the monitor application functionality described in the following Apache Commons page on the: Procrun monitor Application. However, IDM does not include the Procrun service application.

For example, if you've configured IDM as a Windows service, you can start and stop it with the following commands:

```
C:\install-directory\openidm\bin> prunmgr.exe //MR/openidm
C:\install-directory\openidm\bin> prunmgr.exe //MQ/openidm
```

In these commands, `MR` is the option to *Monitor and Run* IDM, and `MQ` stands for *Monitor Quit*, which stops the IDM service.

Start a new project

When you extract the IDM .zip file, you have a default *project* under `/path/to/openidm`.

> **Important**
>
> You can use this project to test customizations, but you should not run the default project in production.

Set up a new project as follows:

1. Create a directory for your new project:

   ```
   mkdir /path/to/my-project
   ```

2. Set up a minimal configuration:

   - If your project will be similar to any of the sample configurations, copy the contents of the sample to your new project.

     For example:

     ```
     cp -r /path/to/openidm/samples/sync-with-ldap/* /path/to/my-project/
     ```

     You can then customize the sample configuration according to your requirements.

   - If you do not want to start with one of the sample configurations, copy the `conf/` and `script/` directories from the default project to your new project directory:

     ```
     cp -pr /path/to/openidm/conf /path/to/my-project/
     cp -pr /path/to/openidm/script /path/to/my-project/
     ```

     You can then customize the default configuration according to your requirements.

3. Start your new project as follows:

   ```
   /path/to/openidm/startup.sh -p /path/to/my-project
   ```

Select a repository

By default, IDM uses an embedded ForgeRock Directory Services (DS) instance for its internal repository. This means that you don't need to install a database to evaluate the software. Before you use IDM in production, you must replace the embedded DS repository with a supported repository. For supported versions, refer to Supported Repositories:

- External DS instance

> **Important**
>
> DS repositories do *not* support storage of audit or workflow data. Do not enable logging to the repository if you are using a DS repository.

- MySQL
- MariaDB

> **Note**
>
> The MySQL repository instructions are also applicable to MariaDB.

- Microsoft SQL
- PostgreSQL
- Oracle Database (Oracle DB)
- IBM DB2 Database

You must also decide how IDM should map objects to the tables in a JDBC database or to organizational units in DS:

- *Generic mapping*, which allows you to store arbitrary objects without special configuration or administration.
- *Explicit mapping*, which maps specific objects and properties to tables and columns in the JDBC database or to organizational units in DS.

By default, IDM uses a generic mapping for user-definable objects, for both a JDBC and a DS repository. A generic mapping speeds up initial deployment, and can make system maintenance more flexible by providing a stable database structure. In a test environment, generic tables let you modify the user and object model easily, without database access, and without the need to constantly add and drop table columns. However, generic mapping does not take full advantage of the underlying database facilities, such as validation within the database and flexible indexing. Using an explicit mapping generally results in a *substantial* performance improvement. It is therefore strongly advised that you change to an explicit mapping before deploying in a production environment. If you are integrating IDM with AM, and using a shared DS repository, you *must* use an explicit schema mapping.

IDM provides a sample configuration, for each JDBC repository, that sets up an explicit mapping for the managed *user* object, and a generic mapping for all other managed objects. This configuration is defined in the files named `/path/to/openidm/db/repository/conf/repo.jdbc-repository-explicit-managed-user.json`. To use this configuration, copy the file that corresponds to your repository to your project's `conf/` directory, and rename it `repo.jdbc.json`. Run the `sample-explicit-managed-user.sql` data definition script (in the `path/to/openidm/db/repository/scripts` directory) to set up the corresponding tables when you configure your JDBC repository.

This section describes how to set up IDM to work with each of the supported repositories, and lists the minimum rights required for database installation and operation.

For information about the repository configuration, refer to Store managed objects in the repository. For more information about generic and explicit mappings, refer to Object mappings.

Embedded DS repository

By default, IDM uses the `conf/repo.ds.json` file to start an embedded DS instance. The embedded DS repository is not supported in production environments.

The embedded DS server uses the embedded DS keystore, and has the following configuration by default:

- `hostname` - `localhost`
- `ldapPort` - `31389`
- `ldapsPort` - `31636`
- `bindDN` - `uid=admin`
- `bindPassword` - `str0ngAdm1nPa55word`
- `adminPort` - `34444`

You can query the embedded repository directly by using the LDAP command-line utilities provided with DS:

This command returns all the objects in the repository of a default IDM project:

```
/path/to/opendj/bin/ldapsearch \
--hostname localhost \
--port 31636 \
--bindDN uid=admin \
--bindPassword str0ngAdm1nPa55word \
--baseDN "dc=openidm,dc=forgerock,dc=com" \
--useSSL \
--trustAll \
"(objectclass=*)"
dn: dc=openidm,dc=forgerock,dc=com
objectClass: domain
objectClass: top
dc: openidm

dn: ou=links,dc=openidm,dc=forgerock,dc=com
objectClass: organizationalUnit
objectClass: top
ou: links

dn: ou=internal,dc=openidm,dc=forgerock,dc=com
objectClass: organizationalUnit
objectClass: top
ou: internal

dn: ou=users,ou=internal,dc=openidm,dc=forgerock,dc=com
objectClass: organizationalUnit
objectClass: top
ou: users
...
```

For more information about the DS command-line utilities, refer to the DS Tools Reference.

To change the administrative port of the embedded DS server, add an `adminPort` property to your project's `conf/repo.ds.json` file before you start IDM. To change any of the other default values, add an `ldapConnectionFactories` property, as shown in the following example.

This excerpt of a `repo.ds.json` sets the administrative port to `4444`. The example changes the bind password to `MyPassw0rd` but shows the structure of the entire `ldapConnectionFactories` property for reference:

```
{
  "embedded": true,
  "maxConnectionAttempts" : 5,
  "adminPort": 4444,
  "ldapConnectionFactories": {
    "bind": {
      "primaryLdapServers": [{ "hostname": "localhost", "port": 31389 }]
    },
    "root": {
      "authentication": {
        "simple": { "bindDn": "uid=admin", "bindPassword": "MyPassw0rd" }
      }
    }
  },
  ...
}
```

It is not necessary to add the entire `ldapConnectionFactories` block to your configuration file, but you must respect the JSON structure. For example, to change only the `hostname`, you would need to add at least the following:

```
{
  ...
  "ldapConnectionFactories": {
    "bind": {
      "primaryLdapServers": [{ "hostname": "my-hostname" }]
    }
  },
  ...
}
```

If you don't include an `ldapConnectionFactories` object, IDM installs an embedded DS server with the default configuration.

External DS repository

IDM supports the following deployment scenarios with a DS repository:

- Single external DS instance
- Two DS instances in an active/passive configuration

  IDM supports two replicated DS instances for backup/availabilty purposes only. Using multiple replicated DS instances as repositories (in a multimaster DS deployment) is not supported.

Configure a single external DS instance as a repository

1. If you have not yet installed DS, download it from the Backstage download site and extract the .zip archive.
2. Install DS according to the instructions in the DS Installation Guide:
   - If you are planning to use a generic object mapping for managed users, install DS with the `idm-repo` profile (see Install DS as an IDM Repository).

- If you are planning to use an explicit object mapping for managed users, install DS with both the `idm-repo` and `am-identity-store` profiles (see Install DS as an IDM Repository and Install DS for AM Identities).

  This example configures DS on the localhost, listening on the following ports:

  - LDAP port: `31389`
  - Admin port: `34444`
  - LDAPS port: `31636`

We've used these ports to avoid a port conflict with the default ports used in the LDAP samples. You can use any host and available ports in the setup. If you use a different host and ports, change the `primaryLdapServers` property in your `repo.ds-external.json` file accordingly.

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

3. In your IDM installation, remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory. For example:

```
cd /path/to/openidm/my-project/conf/
rm repo.ds.json
```

4. Copy the external DS repository configuration file (`repo.ds-external.json`) to your project's `conf` directory and rename it `repo.ds.json` :

```
cp /path/to/openidm/db/ds/conf/repo.ds-external.json my-project/conf/repo.ds.json
```

5. Enable IDM to trust the DS server certificate for your deployment.

   For example, in the default case, where DS servers use TLS key pairs generated using a deploymentId and deploymentIdPassword, import the deploymentId-based CA certificate into the IDM truststore:

```
/path/to/opendj/bin/dskeymgr \
export-ca-cert \
--deploymentId your-deployment-ID \
--deploymentIdPassword password \
--outputFile ds-ca-cert.pem
```

```
keytool \
-importcert \
-alias ds-ca-cert \
-keystore /path/to/openidm/security/truststore \
-storepass:file /path/to/openidm/security/storepass \
-file ds-ca-cert.pem
Owner: CN=Deployment key, O=ForgeRock.com
Issuer: CN=Deployment key, O=ForgeRock.com
...
Trust this certificate? [no]: yes
Certificate was added to keystore
```

6. Adjust the connection settings from IDM to DS in the IDM repository configuration file, `repo.ds.json`:
   - If your DS instance is *not* running on the localhost and listening for LDAP connections on port `31389`, adjust the `primaryLdapServers` property in that file to match your DS setup.
   - Make sure the password for the DS directory superuser (`uid=admin`) matches the DS root user password in the IDM configuration.

   For details about the connection settings, refer to the information in Gateway LDAP Connections in the *DS HTTP User Guide*. (IDM shares these configuration settings with the DS REST to LDAP Gateway.)

7. Start IDM with the configuration for your project. For example:

```
/path/to/openidm/startup.sh -p my-project
Executing ./startup.sh…
Using OPENIDM_HOME:   /path/to/openidm
Using PROJECT_HOME:   /path/to/my-project
Using OPENIDM_OPTS:   -Xmx1024m -Xms1024m
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/my-project/conf/logging.properties
-> OpenIDM version "7.3.2"
OpenIDM ready
```

8. (Optional) Verify that IDM successfully connects to DS:

```
grep 31389 /path/to/opendj/logs/ldap-access.audit.json | tail -n 1 | jq .
{
  "eventName": "DJ-LDAP",
  "client": {
    "ip": "127.0.0.1",
    "port": 35874
  },
  "server": {
    "ip": "127.0.0.1",
    "port": 31389
  },
  "request": {
    "protocol": "LDAP",
    "operation": "SEARCH",
    "connId": 1,
    "msgId": 232,
    "dn": "ou=triggers,ou=scheduler,dc=openidm,dc=forgerock,dc=com",
    "scope": "one",
    "filter": "(&(&(fr-idm-json:caseIgnoreJsonQueryMatch:=/state eq \"NORMAL\")(!(fr-idm-
json:caseIgnoreJsonQueryMatch:=/nodeId pr)))(objectClass=uidObject)(objectClass=fr-idm-generic-obj)
(objectClass=top))",
    "attrs": [
      "objectClass",
      "uid",
      "etag",
      "createTimestamp",
      "modifyTimestamp",
      "fr-idm-json"
    ]
  },
  "transactionId": "transaction-id",
  "response": {
    "status": "SUCCESSFUL",
    "statusCode": "0",
    "elapsedTime": 1,
    "elapsedTimeUnits": "MILLISECONDS",
    "nentries": 0
  },
  "timestamp": "timestamp",
  "_id": "id"
}
```

Configure two DS repositories in an active/passive deployment

With this configuration, IDM fails over to the secondary DS instance if the primary instance becomes unavailable. When the primary DS instance is restarted, that instance again becomes the target of all requests.

1. Download DS from the Backstage download site, and extract the .zip archive.
2. Install two DS servers, according to the instructions in the DS Installation Guide.

> **Important**
>
> When you set up each server, specify a `replicationPort` and `bootstrapReplicationServer` so that both servers are installed as replicas. For information on these setup options, refer to setup in the *DS Tools Reference*.

- If you are planning to use a generic object mapping for managed users, install DS with the `idm-repo` profile (see Install DS as an IDM Repository).
- If you are planning to use an explicit object mapping for managed users, install DS with both the `idm-repo` and `am-identity-store` profiles (see Install DS as an IDM Repository and Install DS for AM Identities).

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

3. In your IDM installation, remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory. For example:

```
cd /path/to/openidm/my-project/conf/
rm repo.ds.json
```

4. Copy the external DS repository configuration file (`repo.ds-external.json`) to your project's `conf` directory and rename it `repo.ds.json` :

```
cp /path/to/openidm/db/ds/conf/repo.ds-external.json my-project/conf/repo.ds.json
```

5. Enable IDM to trust each DS server certificate for your deployment.

   For example, in the default case, where DS servers use TLS key pairs generated using a deploymentId and deploymentIdPassword, import the deploymentId-based CA certificate *for each server* into the IDM truststore.

   You will need to give the CA certificate of the second server a different alias.

```
/path/to/opendj/bin/dskeymgr \
export-ca-cert \
--deploymentId your-deployment-ID \
--deploymentIdPassword password \
--outputFile ds-ca-cert.pem keytool \
-importcert \
-alias ds-ca-cert \
-keystore /path/to/openidm/security/truststore \
-storepass:file /path/to/openidm/security/storepass \
-file ds-ca-cert.pem
Owner: CN=Deployment key, O=ForgeRock.com
Issuer: CN=Deployment key, O=ForgeRock.com
...
Trust this certificate? [no]: yes
Certificate was added to keystore
```

6. Specify the connection settings from IDM to the two DS servers in the `ldapConnectionFactories` property of the repository configuration file (`repo.ds.json`).

   This example assumes that the first DS server runs on the host `ds1.example.com`, and the second DS server runs on the host `ds2.example.com`:

   ```
   "ldapConnectionFactories": {
       "bind": {
           "connectionSecurity": "startTLS",
           "heartBeatIntervalSeconds": 60,
           "heartBeatTimeoutMilliSeconds": 10000,
           "primaryLdapServers": [{ "hostname": "ds1.example.com", "port": 31389 }],
           "secondaryLdapServers": [{ "hostname": "ds2.example.com", "port": 31389 }]
   }
   ```

   Adjust the settings to match your DS server setup.

   For details about the connection settings, refer to the information in Gateway LDAP Connections in the *DS HTTP User Guide*. (IDM shares these configuration settings with the DS REST to LDAP Gateway.)

7. Also in the `repo.ds.json` file, check the `authentication` settings:

   ```
   "root": {
       "inheritFrom": "bind",
       "authentication": {
           "simple": { "bindDn": "uid=admin", "bindPassword": "str0ngAdm1nPa55word" }
       }
   }
   ```

   Make sure that the `bindDn` and `bindPassword` match the bind details of the DS superuser.

8. Start IDM, and verify that the connection to the primary DS server is successful.

9. (Optional) Shut down the primary DS server, and verify that the failover to the secondary server occurs, as expected.

MySQL repository

This procedure assumes that you have installed MySQL on the local host. Follow the MySQL documentation that corresponds to your MySQL version. For supported MySQL versions, refer to Supported Repositories.

Configure IDM to use the new repository *before you start IDM for the first time*. This procedure assumes that a password has already been set for the MySQL root user:

1. Download MySQL Connector/J version 8.0 or later.

   > **Important**
   >
   > Do not use Connector/J versions 8.0.23 through 8.0.25. Why?

2. Unpack the downloaded file, and copy the JAR file to `openidm/bundle/`:

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

3. Make sure that IDM is stopped:

   ```
   /path/to/openidm/shutdown.sh
   OpenIDM is not running, not stopping.
   ```

4. Remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory:

   ```
   rm my-project/conf/repo.ds.json
   ```

5. Copy the MySQL database connection configuration file (`datasource.jdbc-default.json`) and the database table configuration file (`repo.jdbc.json`) to your project's `conf` directory:

   ```
   cp /path/to/openidm/db/mysql/conf/datasource.jdbc-default.json my-project/conf/
   cp /path/to/openidm/db/mysql/conf/repo.jdbc.json my-project/conf/
   ```

6. If you have previously set up a MySQL repository for IDM, you *must* drop the `openidm` database and users before you continue:

   ```
   mysql> drop database openidm;
   Query OK, 21 rows affected (0.63 sec)mysql> drop user openidm;
   Query OK, 0 rows affected (0.02 sec)mysql> drop user openidm@localhost;
   Query OK, 0 rows affected (0.00 sec)
   ```

7. Import the IDM database and tables:

```
cd /path/to/mysql
mysql -u root -p < /path/to/openidm/db/mysql/scripts/openidm.sql
Enter password:
```

> **Note**
>
> If errors like `Access denied for user 'root'@'localhost'` display, and you are deploying on a new installation of Ubuntu 16.04 or later, the `UNIX_SOCKET` plugin may be installed, which applies Linux `root` credentials to MySQL. In that case, substitute `sudo mysql -u root` for `mysql -u root -p` in the commands in this section.

8. Create the IDM database user.

   Run the following script:

   ```
   cd /path/to/mysql
   mysql -u root -p < /path/to/openidm/db/mysql/scripts/createuser.sql
   Enter password:
   ```

   Run the following script:

   ```
   cd /path/to/mysql
   mysql -u root -p < /path/to/openidm/db/mysql/scripts/createuser.mysql57.sql
   Enter password:
   ```

9. Run the script that creates the tables required by the workflow engine:

   ```
   cd /path/to/mysql
   mysql -D openidm -u root -p < /path/to/openidm/db/mysql/scripts/flowable.mysql.all.create.sql
   Enter password:
   ```

10. If you are planning to direct audit logs to this repository, run the script that sets up the audit tables:

    ```
    mysql -D openidm -u root -p < /path/to/openidm/db/mysql/scripts/audit.sql
    Enter password:
    ```

11. Update the connection configuration to reflect your MySQL deployment. The default connection configuration (in the `conf/datasource.jdbc-default.json` file) is:

```
{
    "driverClass" : "com.mysql.cj.jdbc.Driver",
    "jdbcUrl" : "jdbc:mysql://&{openidm.repo.host}:&{openidm.repo.port}/openidm?
allowMultiQueries=true&characterEncoding=utf8&serverTimezone=UTC",
    "databaseName" : "openidm",
    "username" : "openidm",
    "password" : "openidm",
    "connectionTimeout" : 30000,
    "connectionPool" : {
        "type" : "hikari",
        "minimumIdle" : 20,
        "maximumPoolSize" : 50
    }
}
```

Note

- In a production environment, set up SSL as described in the MySQL Connector Developer Guide. The default configuration expects SSL, *which is strongly advised in a production environment*. If you are running this in a test environment, you can bypass the SSL requirement:
  - The default configuration expects SSL, unless you add `&useSSL=false` to the end of the `url`.
  - If you are running MySQL 8.0.11+, add `&allowPublicKeyRetrieval=true` to the end of the `url`.

12. Specify the values for `openidm.repo.host` and `openidm.repo.port` in one of the following ways:

    Set the values in `resolver/boot.properties` or your project's `conf/system.properties` file. For example:

    ```
    openidm.repo.host=localhost
    openidm.repo.port=3306
    ```

    These lines are commented out by default in `resolver/boot.properties`.

    The default MySQL port is `3306`. You can use the `netstat -tlnp` command to check which port your MySQL instance is running on.

    Set the properties in the `OPENIDM_OPTS` environment variable and export that variable before startup. You must include the JVM memory options when you set this variable. For example:

    ```
    export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=3306"
    /path/to/openidm/startup.sh -p my-project
    Executing ./startup.sh...
    Using OPENIDM_HOME:   /path/to/openidm
    Using PROJECT_HOME:   /path/to/openidm
    Using OPENIDM_OPTS:   -Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=3306
    Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
    Using boot properties at /path/to/openidm/resolver/boot.properties
    -> OpenIDM version "7.3.2"
    OpenIDM ready
    ```

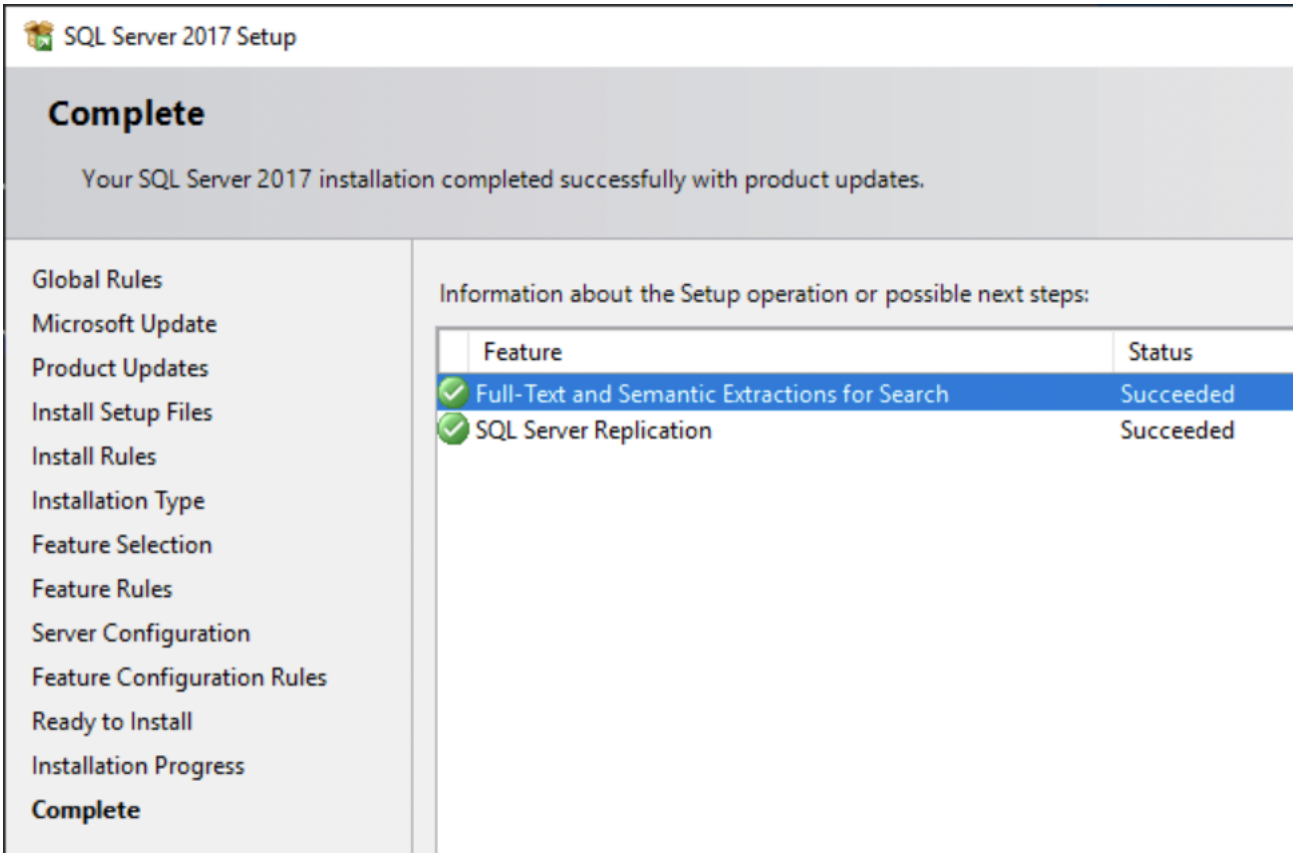
13. Make sure that the server starts without errors.

Microsoft SQL repository

> **Note**
>
> These instructions are specific to Microsoft SQL Server 2017 Evaluation Edition, running on Windows Server 2019, and may require adjustments for other environments.

Install Microsoft SQL Server and associated tools

1. Install Microsoft SQL Server:
    - On the **Installation has completed successfully!** page of the installer, click **Customize** to launch the SQL Server Setup application.
    - Select the server instance you just created, and continue through setup. On the **Feature Selection** step, select *at least* the following options:
        - **SQL Server Replication**
        - **Full-Text and Semantic Extractions for Search**
    - Continue through the setup and verify that the required options were successfully installed, as displayed on the **Complete** page.



2. Download and install SQL Server Management Studio (SSMS).
3. Restart the server.

4. Launch **SSMS**, and connect to the SQL server instance.
5. From the **Object Explorer** , right-click the SQL server instance, and then click **Properties** .
6. On the **Security** page, in the **Server authentication** area, select **SQL Server and Windows Authentication Mode**, and then click **OK**.
7. From the **Object Explorer**, right-click the SQL server instance, and then click **Restart**.
8. Configure TCP/IP:
    - Launch **SQL Server Configuration Manager** .
    - From the left pane, expand the **SQL Server Network Configuration** node, and click **Protocols for serverName**.
    - Double-click **TCP/IP**.
    - In the **TCP/IP Properties** window, from the **Protocol** tab, click the **Enable** drop-down menu, and select **Yes**.
    - Click the **IP Addresses** tab, and make the following changes under **IPAll**, and then click **OK**:
        - In the **TCP Dynamic Ports** field, enter `0` .
        - In the **TCP Port** field, enter `1433` .
    - From the left pane, click **SQL Server Services**, right-click **SQL Server (serverName)**, and then click **Restart**.
    - Configure the firewall to allow IDM to access the SQL Server.

Configure IDM to Use the SQL Repository

1. Install IDM.

> **Warning**
>
> Do not start IDM.

2. Import the IDM data definition language script into Microsoft SQL:
    - Launch **SSMS**.
    - In the **Connect to Server** window, click **Windows Authentication**, and click **Connect**.
    - From the main menu, click **File > Open > File**, navigate to the data definition language script (`C:\path\to\openidm\db\mssql\scripts\openidm.sql`), and click **Open**.
    - Click **Execute**.

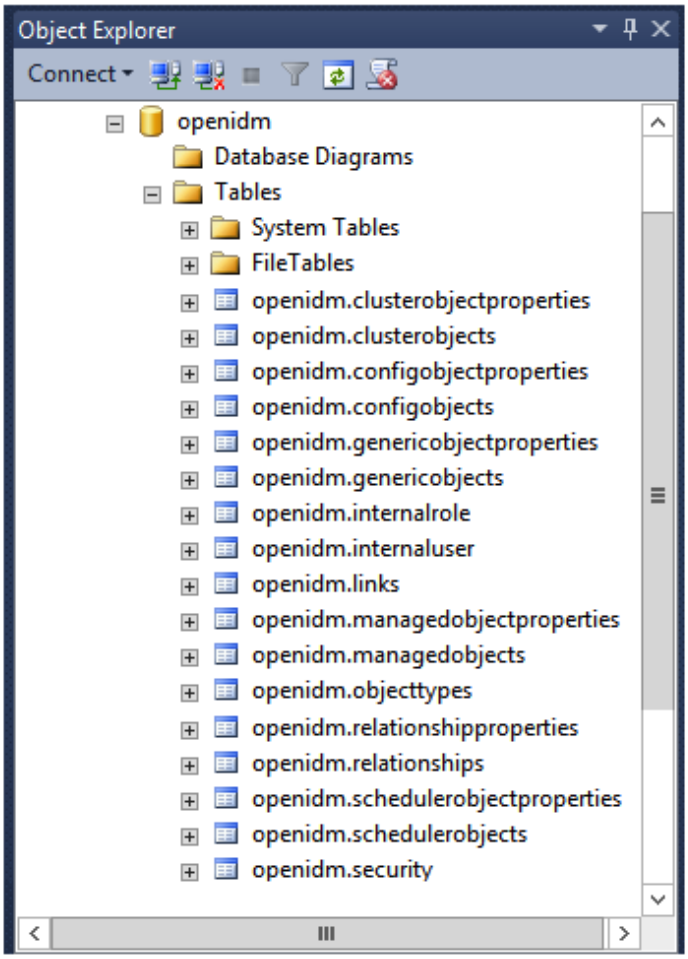
      SSMS displays a message in the **Messages** tab:

```
Commands completed successfully.

Completion time: 2020-11-02709:26:39.1548666-08:00
```

Executing the `openidm.sql` script creates an `openidm` database for use as the internal repository, and an `openidm` user with password `openidm` who has all the required privileges to update the database. You may need to refresh the view in **SSMS** to see the `openidm` database in the **Object Explorer**.

If you expand **Databases > openidm > Tables**, the IDM tables in the `openidm` database display:



3. Execute the script that creates the tables required by the workflow engine. For example:

```
sqlcmd -S localhost -d openidm ^
-i C:\path\to\openidm\db\mssql\scripts\flowable.mssql.all.create.sql
(1 rows affected)
(1 rows affected)
(0 rows affected)
...
```

> **Note**
>
> When you run the `flowable.mssql.all.create.sql` script, the following warning in the log may display:
>
> ```
> Warning! The maximum key length is 900 bytes. The index 'ACT_UNIQ_PROCDEF' has maximum
> length of 1024 bytes. For some combination of large values, the insert/update operation
> will fail.
> ```
>
> It is very unlikely that the key length will be an issue in your deployment, and you can safely ignore this warning.

4. If you are going to direct audit logs to this repository, run the script that sets up the audit tables:

```
sqlcmd -S localhost -d openidm ^
-i C:\path\to\openidm\db\mssql\scripts\audit.sql
```

5. Download the Microsoft JDBC Drivers for SQL Server:
   - Download the JDBC Drivers from Microsoft's download site. IDM requires at least version 7.2 of the driver, which supports OSGi by default.
   - Extract the driver JAR files.
   - Copy the JAR file that corresponds to your Java environment to the `C:\path\to\openidm\bundle` directory. For example:

```
copy mssql-jdbc-7.4.1.jre11.jar C:\path\to\openidm\bundle
```

6. Download the JDBC OSGi Service Package JAR and place it in the `C:\path\to\openidm\bundle` directory.

> **Note**
>
> IDM was tested with version 1.0.0 of the service package.

7. Remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory. For example:

```
cd C:\path\to\openidm\my-project\conf\
del repo.ds.json
```

8. Copy the database connection configuration file for Microsoft SQL (`datasource.jdbc-default.json`) and the database table configuration file (`repo.jdbc.json`) to your project's configuration directory. For example:

```
cd C:\path\to\openidm
copy db\mssql\conf\datasource.jdbc-default.json my-project\conf\
copy db\mssql\conf\repo.jdbc.json my-project\conf\
```

9. Update the connection configuration to reflect your Microsoft SQL deployment. The default connection configuration in the `datasource.jdbc-default.json` file is as follows:

```
{
    "driverClass" : "com.microsoft.sqlserver.jdbc.SQLServerDriver",
    "jdbcUrl" : "jdbc:sqlserver://
&{openidm.repo.host}:&{openidm.repo.port};instanceName=default;databaseName=openidm;applicationName=OpenIDM",
    "databaseName" : "openidm",
    "username" : "openidm",
    "password" : "openidm",
    "connectionTimeout" : 30000,
    "connectionPool" : {
        "type" : "hikari",
        "minimumIdle" : 20,
        "maximumPoolSize" : 50
    }
}
```

Specify the values for `openidm.repo.host` and `openidm.repo.port` in one of the following ways:

Set the values in `resolver/boot.properties` or your project's `conf/system.properties` file. For example:

```
openidm.repo.host=localhost
openidm.repo.port=1433
```

Set the properties in the `OPENIDM_OPTS` environment variable before startup. You must include the JVM memory options when you set this variable. For example:

```
set:OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=1433"
```

10. Start IDM.

Oracle DB repository

Before you set up Oracle DB as the IDM repository, confer with your Oracle DBA to create the database schema, tables, and users. This section assumes that you have configured an Oracle DB with Local Naming Parameters (tnsnames.ora) and a service user for IDM.

> **Important**
>
> IDM supports two connection pools for an Oracle DB:
>
> - Hikari Connection Pool (HikariCP), described in the HikariCP GitHub Repository.
> - Oracle Universal Connection Pool (Oracle UCP), described in the Universal Connection Pool for JDBC Developer's Guide.
>
> Many steps in this procedure will depend on your connection pool type.

Set up Oracle as an IDM repository

1. As the appropriate schema owner, import the IDM schema using the data definition language script (`/path/to/openidm/db/oracle/scripts/openidm.sql`).

2. Use the Oracle SQL Developer Data Modeler to run the script that creates the tables required by the workflow engine:

```
/path/to/openidm/db/oracle/scripts/flowable.oracle.all.create.sql
```

3. If you are planning to direct audit logs to this repository, run the script that sets up audit tables.

   Use the Oracle SQL Developer Data Modeler to run the following script:

```
/path/to/openidm/db/oracle/scripts/audit.sql
```

4. Set the host and port of the Oracle DB instance, either in the `resolver/boot.properties` file or through the `OPENIDM_OPTS` environment variable.

   If you use the `resolver/boot.properties` file, set values for the following variables:

   - `openidm.repo.host = localhost`
   - `openidm.repo.port = 1521`

   If you use the `OPENIDM_OPTS` environment variable, include the JVM memory options when you set the repo host and port. For example:

```
export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=1521"
```

5. Remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory. For example:

```
rm /path/to/openidm/my-project/conf/repo.ds.json
```

6. Copy the Oracle DB repository configuration file (`repo.jdbc.json`) to your project's configuration directory:

```
cp /path/to/openidm/db/oracle/conf/repo.jdbc.json my-project/conf/
```

   Edit the `repo.jdbc.json` file as follows:

```
{
    "dbType" : "ORACLE",
    "useDataSource" : "ucp-oracle",
    ...
}
```

7. Copy the connection configuration file to your project's configuration directory and edit the file for your Oracle DB deployment. The connection configuration file depends on the connection pool that you use:

   1. Copy the following file:

```
cp /path/to/openidm/db/oracle/conf/datasource.jdbc-default.json my-project/conf/
```

Edit the file to reflect your deployment. The default configuration for a HikariCP connection pool is as follows:

```
{
    "driverClass" : "oracle.jdbc.OracleDriver",
    "jdbcUrl" : "jdbc:oracle:thin:@//&{openidm.repo.host}:&{openidm.repo.port}/DEFAULTCATALOG",
    "databaseName" : "openidm",
    "username" : "openidm",
    "password" : "openidm",
    "connectionTimeout" : 30000,
    "connectionPool" : {
        "type" : "hikari",
        "minimumIdle" : 20,
        "maximumPoolSize" : 50
    }
}
```

The `jdbcUrl` corresponds to the URL of the Oracle DB listener, including the service name, based on your configured Local Naming Parameters `tnsnames.ora`. Set this parameter according to your database environment.

The `DEFAULTCATALOG` refers to the SID (system identifier); for example, `orcl`.

The `username` and `password` correspond to the credentials of the service user that connects from IDM.

1. Copy the following file:

```
cp /path/to/openidm/db/oracle/conf/datasource.jdbc-ucp-oracle.json my-project/conf/
```

Edit the file to reflect your deployment. The default connection configuration for an Oracle UCP connection pool is as follows:

```
{
    "databaseName" : "openidm",
    "jsonDataSource" : {
        "class" : "oracle.ucp.jdbc.PoolDataSourceImpl",
        "settings" : {
            "connectionFactoryClassName" : "oracle.jdbc.pool.OracleDataSource",
            "url" : "jdbc:oracle:thin:@&{openidm.repo.host}:&{openidm.repo.port}:SID",
            "user" : "openidm",
            "password" : "openidm",
            "connectionTimeout" : 30000,
            "minPoolSize" : 20,
            "maxPoolSize" : 50
        }
    }
}
```

The `url` corresponds to the URL of the Oracle DB listener, including the service ID (`SID`), based on your configured Local Naming Parameters `tnsnames.ora`. Set this property to the appropriate value for your environment; for example: `jdbc:oracle:thin:@localhost:1521:orcl`.

The `user` and `password` correspond to the credentials of the service user that connects from IDM.

8. Create an OSGi bundle for the Oracle DB driver, as follows:

   ◦ Download the JDBC drivers for your Oracle DB version.

   The files that you download depend on your Oracle DB version, and on whether you are using HikariCP or Oracle UCP. Because the version numbers change with minor updates, you must search for the precise corresponding files on `oracle.com`:

   - Download the `ojdbc*.jar` file that corresponds to your Oracle DB version.
   - Download the most recent `bnd` JAR file from https://repo1.maven.org/maven2/biz/aQute/bnd/biz.aQute.bnd/. The bnd utility lets you create OSGi bundles for JDBC libraries that do not yet support OSGi.

     1. Download the following additional files:

        - `ucp.jar`
        - `ons.jar`

   Copy the downloaded files to the `/path/to/openidm/db/oracle/scripts` directory.

   ◦ The `/path/to/openidm/db/oracle/scripts` directory includes an `ojdbc8.bnd` file that specifies the version information for your JDBC driver.

   Edit the driver version in that file if necessary. The default file is as follows:

   ```
   version=12.2.0.1
   Export-Package: *;version=${version}
   Bundle-Name: Oracle Database 12.2.0.1 JDBC Driver
   Bundle-SymbolicName: oracle.jdbc.OracleDriver
   Bundle-Version: ${version}
   Import-Package: *;resolution:=optional
   ```

   Note

   - Do not include trailing zeros in the version number. For example, for Oracle 12.2.0.1.0, set the version string to `version=12.2.0.1`.
   - Oracle DB 12cR2 (12.2.0.1) uses the drivers in `ojdbc8.jar`.

   ◦ From the `/path/to/openidm/db/oracle/scripts` directory, run the following command to create the OSGi bundle, replacing the `*` with your Oracle DB driver version:

   ```
   java -jar biz.aQute.bnd-version.jar wrap --properties ojdbc*.bnd --output ojdbc*-osgi.jar
   ojdbc*.jar
   ```

     1. Create `bnd` files for the `ucp.jar` and `ons.jar` files. The following examples assume version 12.2.0 Oracle JDBC drivers:

        - `ucp.bnd`

```
version=12.2.0
Export-Package: oracle.ucp.*;version=${version}
Bundle-Name: Oracle Universal Connection Pool
Bundle-SymbolicName: oracle.ucp
Bundle-Version: ${version}
Import-Package: *;resolution:=optional
DynamicImport-Package: *
```

- `ons.bnd`

```
version=12.2.0
Export-Package: *;version=${version}
Bundle-Name: Oracle ONS
Bundle-SymbolicName: oracle.ons
Bundle-Version: ${version}
Import-Package: *;resolution:=optional
```

Save the `bnd` files in the `/path/to/openidm/db/oracle/scripts` directory, then run the following commands to create the corresponding OSGi bundles:

```
cd /path/to/openidm/db/oracle/scripts
java -jar biz.aQute.bnd-version.jar wrap --properties ucp.bnd --output ucp-osgi.jar
ucp.jar
java -jar biz.aQute.bnd-version.jar wrap --properties ons.bnd --output ons-osgi.jar
ons.jar
```

You can ignore any `private references` warnings that are logged when you build these bundles.

- Move all the OSGi bundle files to the `openidm/bundle` directory.

9. When you have set up Oracle DB for use as the internal repository, make sure that the server starts without errors.

PostgreSQL repository

- Configure the PostgreSQL Repository and IDM
- Configure Searchable Array Fields

Configure the PostgreSQL repository and IDM

This procedure assumes that a supported version of PostgreSQL is installed and running on the local host. *Before starting IDM for the first time*, configure the server to use a PostgreSQL repository.

Note

The `path/to/openidm/db/postgresql/scripts/createuser.pgsql` script creates an `openidm` database and role, with a default password of `openidm`. The script also grants the appropriate permissions.
Edit this script if you want to change the password of the `openidm` role, for example:

```
create database openidm encoding 'utf8';
create role openidm with LOGIN NOSUPERUSER NOCREATEDB NOCREATEROLE inherit password 'newPassword';
grant all privileges on database openidm to openidm;
```

1. Edit the Postgres client authentication configuration file, `pg_hba.conf`. Add the following entries for the following users: `postgres` and `openidm`:

```
local   all             openidm                                 trust
local   all             postgres                                trust
```

2. As the `postgres` user, execute the `createuser.pgsql` script as follows:

```
psql -U postgres < /path/to/openidm/db/postgresql/scripts/createuser.pgsql
CREATE DATABASE
CREATE ROLE
GRANT
```

3. Run the `openidm.pgsql` script as the new `openidm` user that you created in the first step:

```
psql -U openidm < /path/to/openidm/db/postgresql/scripts/openidm.pgsql
CREATE SCHEMA
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE INDEX
CREATE INDEX
...
START TRANSACTION
INSERT 0 1
INSERT 0 1
COMMIT
CREATE INDEX
CREATE INDEX
```

Your database has now been initialized.

4. Run the script that creates the tables required by the workflow engine:

```
psql -d openidm -U openidm < /path/to/openidm/db/postgresql/scripts/flowable.postgres.all.create.sql
```

5. If you plan to direct audit logs to this repository, run the script that sets up the audit tables:

```
psql -d openidm -U openidm < /path/to/openidm/db/postgresql/scripts/audit.pgsql
```

6. Remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory. For example:

```
cd /path/to/openidm/my-project/conf/
rm repo.ds.json
```

7. Copy the database connection configuration file for PostgreSQL (`datasource.jdbc-default.json`) and the database table file (`repo.jdbc.json`) to your project's configuration directory. For example:

```
cp /path/to/openidm/db/postgresql/conf/datasource.jdbc-default.json my-project/conf/
cp /path/to/openidm/db/postgresql/conf/repo.jdbc.json my-project/conf/
```

8. Update the connection configuration to reflect your PostgreSQL deployment. The default connection configuration in the `datasource.jdbc-default.json` file is as follows:

```
{
    "driverClass" : "org.postgresql.Driver",
    "jdbcUrl" : "jdbc:postgresql://&{openidm.repo.host}:&{openidm.repo.port}/openidm",
    "databaseName" : "openidm",
    "username" : "openidm",
    "password" : "openidm",
    "connectionTimeout" : 30000,
    "connectionPool" : {
        "type" : "hikari",
        "minimumIdle" : 20,
        "maximumPoolSize" : 50
    }
}
```

If you changed the password in step 1 of this procedure, edit the `datasource.jdbc-default.json` file to set the value for the `password` field to whatever password you set for the `openidm` user.

Specify the values for `openidm.repo.host` and `openidm.repo.port` in one of the following ways:

Set the values in your `resolver/boot.properties` file:

```
openidm.repo.host = localhost
openidm.repo.port = 5432
```

Set the properties in the `OPENIDM_OPTS` environment variable and export that variable before startup. You must include the JVM memory options when you set this variable. For example:

```
export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=5432"
/path/to/openidm/startup.sh -p my-project
Executing ./startup.sh…
Using OPENIDM_HOME:   /path/to/openidm
Using PROJECT_HOME:   /path/to/openidm
Using OPENIDM_OPTS:   -Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=5432
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
Using boot properties at /path/to/openidm/resolver/boot.properties
-> OpenIDM version "7.3.2"
OpenIDM ready
```

9. PostgreSQL is now set up for use as the internal repository. Make sure that the server starts without errors.
10. Set up indexes to tune the PostgreSQL repository according to your specific deployment.

> **Important**
>
> No indexes are set by default. If you do not tune the repository correctly by creating the required indexes, the performance of your service can be severely impacted. For example, setting too many indexes can have an adverse effect on performance during managed object creation. Conversely, not indexing fields that are searched will severely impact search performance.

IDM includes a `/path/to/openidm/db/postgresql/scripts/default_schema_optimization.pgsql` script that sets up a number of indexes. This script includes extensive comments on the indexes that are being created. Review the script before you run it to ensure that all the indexes are suitable for your deployment.

When you have refined the script for your deployment, execute the script as a user with superuser privileges, so that the required extensions can be created. By default, this is the `postgres` user:

```
psql -U postgres openidm < /path/to/openidm/db/postgresql/scripts/default_schema_optimization.pgsql
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE INDEX
```

Configure Array Fields

Optionally, you can configure arrays in order to perform REST queries. This feature only works for `genericMapping` objects.

1. Edit the `repo.jdbc.json` file to identify which fields/properties are stored as `JSON_LIST` arrays:

```
"genericMapping" : {
    "managed/*" : {
        "mainTable" : "managedobjects"
    },
    "managed/role" : {
        "mainTable": "managedobjects",
        "properties" : {
            "/stringArrayField" : {
                "type" : "JSON_LIST"
            }
        }
    },
...
```

2. Edit the `managed.json` file and add the properties to `role`:

```
"stringArrayField" : {
    "description" : "An array of strings",
    "title" : "String array field",
    "viewable" : true,
    "returnByDefault" : false,
    "type" : "array",
    "items" : {
        "type" : "string",
        "title" : "Some strings"
    }
}
```

IBM DB2 repository

This section makes the following assumptions about the DB2 environment. If these assumptions do not match your DB2 environment, adapt the subsequent instructions accordingly.

- DB2 is running on the localhost, and is listening on the default port (50000).
- The user `db2inst1` is configured as the DB2 instance owner, and has the password `Passw0rd1`.

This section assumes that you will use basic username/password authentication. You can also configure Kerberos authentication with a DB2 repository.

Before you start, make sure that the server is stopped.

```
/path/to/openidm/shutdown.sh
OpenIDM is not running, not stopping.
```

Configure IDM to use the DB2 repository, as described in the following steps:

1. Create an OSGi bundle for the DB2 JDBC driver, as follows:
    - Download the DB2 JDBC driver for your database version from the IBM download site and place it in the `openidm/db/db2/scripts` directory.

Use either the `db2jcc.jar` or `db2jcc4.jar`, depending on your DB2 version. For more information, refer to the DB2 JDBC Driver Versions.

```
ls /path/to/db/db2/scripts/
db2jcc.jar  openidm.sql
```

- Create a `bnd` file and edit it to match the version information for your JDBC driver.

  You can use the sample `bnd` file located in `openidm/db/mssql/scripts`. Copy that file to the directory with the JDBC driver:

  ```
  cd /path/to/openidm/db
  cp mssql/scripts/sqljdbc4.bnd db2/scripts/
  ls db2/scripts
  db2jcc.jar  openidm.sql  sqljdbc4.bnd
  ```

  The JDBC driver version information for your driver is located in the `Specification-Version` property in the MANIFEST file of the driver.

  ```
  cd /path/to/openidm/db/db2/scripts
  unzip -q -c db2jcc.jar META-INF/MANIFEST.MF
  Manifest-Version: 1.0
  Created-By: 1.4.2 (IBM Corporation)
  ```

  Edit the `bnd` file to match the JDBC driver version:

  ```
  **more sqljdbc4.bnd**
  ...
  version=1.0
  Export-Package: *;version=${version}
  Bundle-Name: IBM JDBC DB2 Driver
  Bundle-SymbolicName: com.ibm.db2.jcc.db2driver
  Bundle-Version: ${version}
  ```

- Download the most recent `bnd` JAR file from https://repo1.maven.org/maven2/biz/aQute/bnd/biz.aQute.bnd/. The bnd utility lets you create OSGi bundles for JDBC libraries that do not yet support OSGi.

  Place the `bnd` JAR file in the same directory as the JDBC driver:

  ```
  ls /path/to/openidm/db/db2/scripts
  biz.aQute.bnd-version.jar db2jcc.jar
  ```

- Change to the directory in which the script files are located and run the following command to create the OSGi bundle:

```
cd /path/to/openidm/db/db2/scripts
java -jar biz.aQute.bnd-version.jar wrap --properties sqljdbc4.bnd --output db2jcc-osgi.jar
db2jcc.jar
```

This command creates an OSGi bundle, as defined by the `--output` option: `db2jcc-osgi.jar`:

```
ls /path/to/openidm/db/db2/scripts
biz.aQute.bnd-version.jar db2jcc-osgi.jar  db2jcc.jar
```

- Move the OSGi bundle fle to the `openidm/bundle` directory:

```
mv db2jcc-osgi.jar /path/to/openidm/bundle/
```

2. Remove the default DS repository configuration file (`repo.ds.json`) from your project's `conf/` directory. For example:

```
cd /path/to/openidm/my-project/conf/
rm repo.ds.json
```

3. Copy the database connection configuration file for DB2 (`datasource.jdbc-default.json`) and the database table configuration file (`repo.jdbc.json`) to your project's configuration directory. For example:

```
cp /path/to/openidm/db/db2/conf/datasource.jdbc-default.json my-project/conf/
cp /path/to/openidm/db/db2/conf/repo.jdbc.json my-project/conf/
```

4. Update the connection configuration to reflect your DB2 deployment. The default connection configuration in the `datasource.jdbc-default.json` file is as follows:

```
{
    "driverClass" : "com.ibm.db2.jcc.DB2Driver",
    "jdbcUrl" : "jdbc:db2://&{openidm.repo.host}:&{openidm.repo.port}/
dopenidm:retrieveMessagesFromServerOnGetMessage=true;",
    "databaseName" : "sopenidm",
    "username" : "openidm",
    "password" : "openidm",
    "connectionTimeout" : 30000,
    "connectionPool" : {
        "type" : "hikari",
        "minimumIdle" : 20,
        "maximumPoolSize" : 50
    }
}
```

Specify the values for `openidm.repo.host` and `openidm.repo.port` in one of the following ways:

Set the values in `resolver/boot.properties` or your project's `conf/system.properties` file, for example:

```
openidm.repo.host = localhost
openidm.repo.port = 50000
```

Set the properties in the `OPENIDM_OPTS` environment variable and export that variable before startup. You must include the JVM memory options when you set this variable. For example:

```
export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=50000"
/path/to/openidm/startup.sh -p my-project
Executing ./startup.sh...
Using OPENIDM_HOME:   /path/to/openidm
Using PROJECT_HOME:   /path/to/openidm
Using OPENIDM_OPTS:   -Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -Dopenidm.repo.port=50000
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
Using boot properties at /path/to/openidm/resolver/boot.properties
-> OpenIDM version "7.3.2"
OpenIDM ready
```

5. Create a user database for IDM (`dopenidm`).

```
db2 create database dopenidm
```

6. Import the IDM data definition language script into your DB2 instance.

```
cd /path/to/openidm
db2 -i -tf db/db2/scripts/openidm.sql
```

The database schema is defined in the `SOPENIDM` database.

7. You can show the list of tables in the repository, using the `db2 list` command, as follows:

```
db2 LIST TABLES for all
Table/View                      Schema          Type  Creation time
------------------------------- --------------- ----- --------------------------
CLUSTEROBJECTPROPERTIES         SOPENIDM        T     2015-10-01-11.58.05.968933
CLUSTEROBJECTS                  SOPENIDM        T     2015-10-01-11.58.05.607075
CONFIGOBJECTPROPERTIES          SOPENIDM        T     2015-10-01-11.58.01.039999
CONFIGOBJECTS                   SOPENIDM        T     2015-10-01-11.58.00.570231
GENERICOBJECTPROPERTIES         SOPENIDM        T     2015-10-01-11.57.59.583530
GENERICOBJECTS                  SOPENIDM        T     2015-10-01-11.57.59.152221
INTERNALUSER                    SOPENIDM        T     2015-10-01-11.58.04.060990
LINKS                           SOPENIDM        T     2015-10-01-11.58.01.349194
MANAGEDOBJECTPROPERTIES         SOPENIDM        T     2015-10-01-11.58.00.261556
MANAGEDOBJECTS                  SOPENIDM        T     2015-10-01-11.57.59.890152
...
```

8. Connect to the `openidm` database, and run the script that creates the tables required by the workflow engine:

```
db2 connect to dopenidm
db2 -i -tf /path/to/openidm/db/db2/scripts/flowable.db2.all.create.sql
```

9. If you plan to direct audit logs to this repository, run the script that sets up the audit tables:

```
db2 -i -tf /path/to/openidm/db/db2/scripts/audit.sql
```

When you have set up DB2 for use as the internal repository, make sure that the server starts without errors.

Kerberos authentication with a DB2 repository

By default, IDM uses the username and password configured in the repository connection configuration file (`conf/datasource.jdbc-default.json`) to connect to the DB2 repository. You can configure IDM to use Kerberos authentication instead.

In this scenario, IDM acts as a *client* and requests a Kerberos ticket for a *service*, which is DB2, through the JDBC driver.

This section assumes that you have configured DB2 for Kerberos authentication. If that is not the case, follow the instructions in the corresponding DB2 documentation before you read this section.

The following diagram shows how the ticket is obtained and how the keytab is referenced from IDM's `jaas.conf` file.

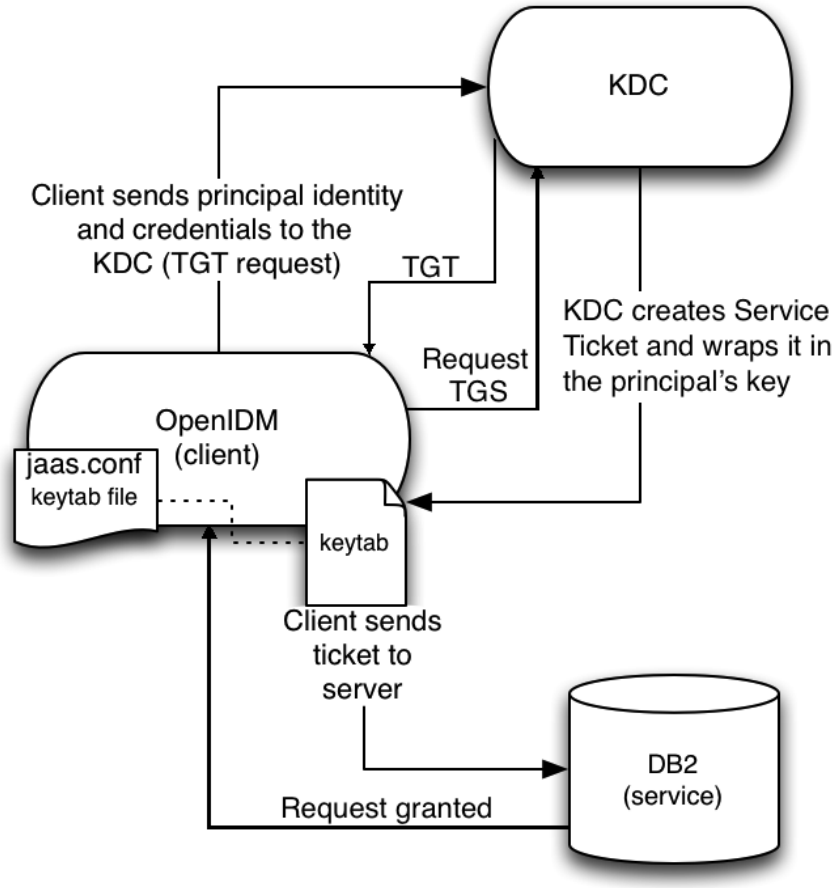


Figure 1. Using Kerberos to Connect to a DB2 Repository

1. Create a keytab file, specifically for use by IDM.

   A Kerberos keytab file (`krb5.keytab`) is an encrypted copy of the host's key. The keytab enables DB2 to validate the Kerberos ticket that it receives from IDM. You must create a keytab file on the host that IDM runs on. The keytab file must be secured in the same way that you would secure any password file. Specifically, only the user running IDM should have read and write access to this file.

   Create a keytab for DB2 authentication, in the file `openidm/security/idm.keytab/` :

   ```
   kadmin -p kadmin/admin -w password
   kadmin: ktadd -k /path/to/openidm/security/idm.keytab db2/idm.example.com
   ```

2. Make sure that the DB2 user has read access to the keytab.

3. Copy the DB2 Java Authentication and Authorization Service (JAAS) configuration file to the IDM `security` directory:

   ```
   cp /path/to/openidm/db/db2/conf/jaas.conf /path/to/openidm/security/
   ```

   By default, IDM assumes that the keytab is in the file `openidm/security/idm.keytab` and that the principal identity is `db2/idm.example.com@EXAMPLE.COM` . Change the following lines in the `jaas.conf` file if you are using a different keytab:

```
keyTab="security/idm.keytab"
principal="db2/idm.example.com@EXAMPLE.COM"
```

4. Adjust the authentication details in your DB2 connection configuration file (`conf/datasource.jdbc-default.json`) to remove the `password` field and change the username to the instance owner (`db2`). The following excerpt shows the modified file:

```
{
    ...
    "databaseName" : "sopenidm",
    "username" : "db2",
    "connectionTimeout" : 30000,
    ...
}
```

5. Edit your project's `conf/system.properties` file, to add the required Java options for Kerberos authentication.

   In particular, add the following two lines to that file:

```
db2.jcc.securityMechanism=11
java.security.auth.login.config=security/jaas.conf
```

6. Restart IDM.

JDBC repository configuration

JDBC database access rights

> **Note**
>
> The following topic only applies if you have set up a JDBC repository, as described in Select a repository.

In general, IDM requires minimal access rights to the JDBC repository for daily operation. This section lists the minimum permissions required, and suggests a strategy for restricting database access in your deployment.

The JDBC repository used by IDM requires only one *relevant* user — the service account that is used to create the tables. Generally, the details of this account are configured in the repository connection file (`datasource.jdbc-default.json`). By default, the username and password for this account are `openidm` and `openidm` , regardless of the database type.

All other users are created by the `db/database-type/scripts/openidm.sql` script. The `openidm` user account must have SELECT, UPDATE, INSERT, and DELETE permissions on all the `openidm` tables that are created by this script, by the scripts that create the tables specific to the Flowable workflow engine, and by the script that sets up the audit tables if you are using the repository audit event handler.

Case insensitivity for a JDBC repo

> **Note**
>
> The following topic only applies if you have set up a JDBC repository, as described in Select a repository

A DS repository is case-insensitive by default. The supported JDBC repositories are generally case-sensitive by default. Case-sensitivity can cause issues if queries expect results to be returned, regardless of case.

For example, with the default configuration of a MySQL database, a search for an email address of `scarter@example.com` might return a result, while a search for `scarter@EXAMPLE.COM` might return an `Unable to find account` error.

If you need to support case-insensitive queries, you must configure a case-insensitive collation in your JDBC repository, on the specific columns that require it. For example:

- For a generic managed object mapping in MySQL or MariaDB, change the default collation of the `managedobjectproperties.propvalue` column to `utf8_general_ci`. Note that this changes case-sensitivity for *all* managed object properties. To change case-sensitivity for all the properties of a specific object, specify a different table for the `propertiesTable` entry in your `repo.jdbc.json` for that object, and adjust the collation on that table. To change case-sensitivity only for certain properties of an object, use an explicit mapping.
- For a PostgreSQL repository, use an explicit table structure if you require case-insensitivity. Managing case-insensitivity at scale with generic tables in PostgreSQL is not supported. For more information about object mappings, refer to Mappings with a JDBC repository.
- For an Oracle DB repository, refer to the corresponding Oracle documentation.
- For a SQL Server repository, refer to the corresponding Windows documentation.
- For a DB2 repository, refer to the corresponding DB2 documentation.

JDBC over SSL

> **Note**
>
> The following topic only applies if you have set up a JDBC repository, as described in Select a repository

This procedure assumes that you have already set up your JDBC repository, as described in the previous sections. The exact steps to connect to a JDBC repository over SSL depend on your repository. This procedure describes the steps for a MySQL 8 repository. If you are using a different JDBC repository, use the corresponding documentation for that repository, and adjust the steps accordingly.

1. Change the `jdbcUrl` property in your repository connection configuration file (`conf/datasource.jdbc-default.json`).

   The exact value of the `jdbcUrl` property will depend on your JDBC database, and on the version of your JDBC driver:

   ```
   "jdbcUrl" : "jdbc:mysql://&{openidm.repo.host}:&{openidm.repo.port}/openidm?
   allowMultiQueries=true&characterEncoding=utf8&useSSL=true&verifyServerCertificate=true&requireSSL=true"
   ```

```
"jdbcUrl" : "jdbc:mysql://&{openidm.repo.host}:&{openidm.repo.port}/openidm?
allowMultiQueries=true&characterEncoding=utf8&sslMode=VERIFY_CA&requireSSL=true"
```

> **Note**
>
> For Azure MySQL, JDBC Driver Version 8.0.17+ is **required**.

2. Create and verify the SSL certificate and key files required to support encrypted connections to the JDBC repository.

   For MySQL 8, use one of the procedures in the MySQL docs.

3. Configure the JDBC repository to use encrypted connections.

   For MySQL 8, follow the MySQL docs.

4. Check that the connection to the database is over SSL by running a command similar to the following:

```
mysql -u root -P 3306 -p
mysql> show variables like "%have_ssl%";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_ssl      | YES   |
+---------------+-------+
1 row in set (0.00 sec)
```

5. Convert your MySQL client key and certificate files to a PKCS #12 archive. For example:

```
openssl pkcs12 -export \
-in client-cert.pem \
-inkey client-key.pem \
-name "mysqlclient" \
-passout pass:changeit \
-out client-keystore.p12
```

6. Import the `client-keystore.p12` into the IDM keystore:

```
keytool \
-importkeystore \
-srckeystore client-keystore.p12 \
-srcstoretype pkcs12 \
-srcstorepass changeit \
-destkeystore /path/to/openidm/security/keystore.jceks \
-deststoretype jceks \
-deststorepass changeit
```

> **Important**
>
> For AWS RDS MySQL **and** Azure MySQL, no client certificates are provided. In this case, you must create an empty keystore for client certificates, and add the following to the `jdbcUrl` property in your repository connection configuration file (`conf/datasource.jdbc-default.json`):
>
> ```
> &clientCertificateKeyStoreUrl=file:/opt/idm/security/
> empty.jks&clientCertificateKeyStorePassword=changeit
> ```

7. Import your MySQL CA certificate into the IDM truststore.

   ```
   keytool \
   -importcert \
   -trustcacerts \
   -file ca-cert.pem \
   -alias "DB cert" \
   -keystore /path/to/openidm/security/truststore
   ```

   You are prompted for a keystore password. You must use the same password as is shown in your `resolver/boot.properties` file. The default truststore password is:

   ```
   openidm.truststore.password=changeit
   ```

   After entering a keystore password, you are prompted with the following question. Assuming you have included an appropriate `ca-cert.pem` file, enter `yes`.

   ```
   Trust this certificate? [no]:
   ```

8. Open your project's `conf/system.properties` file. Add the following line to that file. If appropriate, substitute the path to your own truststore:

   ```
   # Set the truststore
   javax.net.ssl.trustStore=&{idm.install.dir}/security/truststore
   ```

   Even if you are setting up this instance of IDM as part of a cluster, you must configure this initial truststore. After this instance joins a cluster, the SSL keys in this particular truststore are replaced.

Configuration and monitoring

Startup configuration

By default, IDM starts with the configuration, script, and binary files in the `openidm/conf` , `openidm/script` , and `openidm/bin` directories. You can launch IDM with a different set of configuration, script, and binary files for test purposes, to manage different projects, or to run one of the included samples.

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

`-p | --project-location {/path/to/project/directory}`

The project location specifies the directory that contains the configuration and script files that IDM will use.

All configuration objects and any artifacts that are not in the bundled defaults (such as custom scripts) *must* be included in the project location. These objects include all files otherwise included in the `openidm/conf` and `openidm/script` directories.

For example, the following command starts the server with the configuration of the `sync-with-csv` sample (located in `/path/to/openidm/samples/sync-with-csv`):

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

If you do not provide an absolute path, the project location path is relative to the system property, `user.dir`. IDM sets `idm.instance.dir` to that relative directory path. Alternatively, if you start the server without the `-p` option, IDM sets `idm.instance.dir` to `/path/to/openidm`.

> **Note**
>
> In this documentation, "your project" refers to the value of `idm.instance.dir`.

`-w |--working-location {/path/to/working/directory}`

The working location specifies the directory in which the embedded DS instance is installed, and the directory to which IDM writes its database cache, audit logs, and Felix cache. The working location includes everything that is in the default `db/`, `audit/`, and `felix-cache/` subdirectories.

The following command specifies that IDM writes its database cache and audit data to `/Users/admin/openidm/storage`:

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

If you do not provide an absolute path, the path is relative to the system property, `user.dir`. IDM sets `idm.data.dir` to that relative directory path. If you do not specify a working location, IDM sets `idm.data.dir` to `/path/to/openidm`. This means the default working location data is located in the `openidm/db`, `openidm/felix-cache`, and `openidm/audit` directories.

Note that this property does not affect the location of the IDM system logs. To change the location of these logs, edit the `conf/logging.properties` file.

You can also change the location of the Felix cache, by editing the `conf/config.properties` file, or by starting the server with the `-s` option, described later in this section.

`-c | --config {/path/to/config/file}`

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

Unless you are working with a highly customized deployment, you should not modify the default framework configuration.

`-P {property=value}`

Any properties passed to the startup script with the `-P` option are used when the server loads the `launcher.json` startup configuration file.

Options specified here have the lowest order of precedence when the configuration is loaded. If the same property is defined in any other configuration source, the value specified here is ignored.

`-s | --storage {/path/to/storage/directory}`

Specifies the OSGi storage location of the cached configuration files.

You can use this option to redirect output if you are installing on a read-only filesystem volume, or if you are testing different configurations. Sometimes, when you start the server with two different configurations, one after the other, the cached configurations are merged and cause problems. Specifying a storage location puts the cached configuration files in that location, and avoids conflicts with cached files from previous startups.

Additionally, IDM sets the system property `idm.install.dir` to the location IDM is installed in. For example, if IDM was installed in `/Users/admin/openidm/`, that is what `idm.install.dir` will be set to.

For information about changing the startup configuration by substituting property values, refer to Property Value Substitution.

Monitor server health

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

The health check service lets you monitor the status of internal resources.

To monitor the status of external resources, such as LDAP servers and external databases, use the commands described in Check external system status over REST.

Basic health checks

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

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

To verify the current server state, use the following REST call:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/info/ping"
{
  "_id": "",
  "_rev": "",
  "shortDesc": "OpenIDM ready",
  "state": "ACTIVE_READY"
}
```

Session information

To obtain information about the current IDM session, beyond basic health checks, use the following REST call:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/info/login"
{
  "_id": "login",
  "authenticationId": "openidm-admin",
  "authorization": {
    "userRolesProperty": "authzRoles",
    "component": "internal/user",
    "authLogin": false,
    "authenticationIdProperty": "username",
    "roles": [
      "internal/role/openidm-admin",
      "internal/role/openidm-authorized"
    ],
    "ipAddress": "0:0:0:0:0:0:0:1",
    "authenticationId": "openidm-admin",
    "id": "openidm-admin",
    "moduleId": "INTERNAL_USER",
    "queryId": "credential-internaluser-query"
  }
}
```

> **Note**
>
> The precise output of this command depends on the authentication module responsible for authenticating the user. For more information about authentication modules, refer to Authentication and Session Modules

Health check service

The configurable health check service verifies the status of the modules and services required for an operational system. During system startup, IDM checks that these modules and services are available and reports on any requirements that have not been met. If dynamic configuration changes are made, IDM rechecks that the required modules and services are functioning, to allow ongoing monitoring of system operation.

IDM checks all required modules. Examples of those modules are shown here:

```
"org.forgerock.openicf.framework.connector-framework"
"org.forgerock.openicf.framework.connector-framework-internal"
"org.forgerock.openicf.framework.connector-framework-osgi"
"org.forgerock.openidm.audit"
"org.forgerock.openidm.core"
"org.forgerock.openidm.enhanced-config"
"org.forgerock.openidm.external-email"
...
"org.forgerock.openidm.system"
"org.forgerock.openidm.ui"
"org.forgerock.openidm.util"
"org.forgerock.commons.org.forgerock.json.resource"
"org.forgerock.commons.org.forgerock.util"
"org.forgerock.openidm.security-jetty"
"org.forgerock.openidm.jetty-fragment"
"org.forgerock.openidm.quartz-fragment"
"org.ops4j.pax.web.pax-web-extender-whiteboard"
"org.forgerock.openidm.scheduler"
"org.ops4j.pax.web.pax-web-jetty"
"org.forgerock.openidm.repo-jdbc"
"org.forgerock.openidm.repo-ds"
"org.forgerock.openidm.config"
"org.forgerock.openidm.crypto"
```

IDM checks all required services. Examples of those services are shown here:

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

You can replace the list of required modules and services, or add to it, by adding the following lines to your `resolver/boot.properties` file. Bundles and services are specified as a list of symbolic names, separated by commas:

- `openidm.healthservice.reqbundles` - overrides the default required bundles.
- `openidm.healthservice.reqservices` - overrides the default required services.
- `openidm.healthservice.additionalreqbundles` - specifies required bundles (in addition to the default list).
- `openidm.healthservice.additionalreqservices` - specifies required services (in addition to the default list).

Note

By default, the server is given 15 seconds to start up all the required bundles and services before system readiness is assessed. This is not the total start time, but the time required to complete the service startup after the framework has started. You can change this default by setting the value of the `servicestartmax` property (in milliseconds) in your `resolver/boot.properties` file. This example sets the startup time to five seconds:

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

Installed modules and features

You can query the enabled features over REST at the `info/features` endpoint. The feature availability service determines the set of possible features from the active bundles and provides the following information:

- The name and `_id` of the feature
- Whether the feature is enabled
- If the feature is enabled, the REST endpoint on which that feature can be accessed

Example

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/info/features?_queryFilter=true"
{
  "result": [
    {
      "_id": "retrieveUsername",
      "name": "retrieveUsername",
      "enabled": false,
      "endpoints": []
    },
    {
      "_id": "identityProviders",
      "name": "identityProviders",
      "enabled": true,
      "endpoints": [
        "identityProviders"
      ]
    },
    {
      "_id": "workflow",
      "name": "workflow",
      "enabled": true,
      "endpoints": [
        "workflow*"
      ]
    },
    {
      "_id": "passwordReset",
      "name": "passwordReset",
      "enabled": false,
      "endpoints": []
    },
    {
      "_id": "registration",
      "name": "registration",
      "enabled": true,
      "endpoints": [
        "selfservice/registration"
      ]
    },
    {
      "_id": "email",
      "name": "email",
      "enabled": false,
```

```
        "endpoints": []
      }
    ],
    ...
  }
```

IDM in a cluster

To ensure that your identity management service remains available in the event of system failure, you can deploy multiple IDM instances in a cluster. In a clustered environment, each instance points to the same external repository.

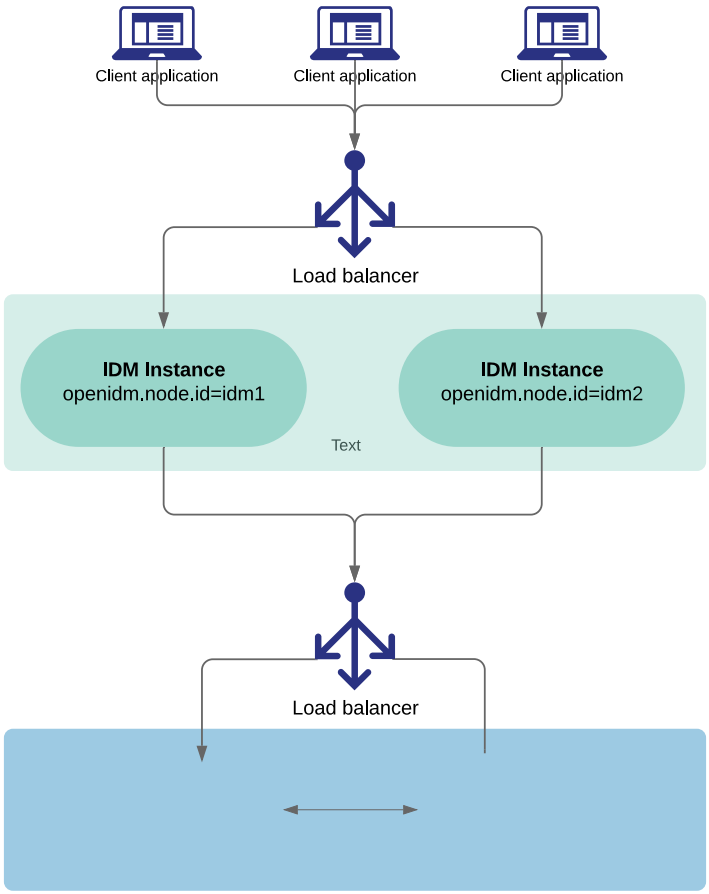
If one instance in a cluster shuts down or fails to check in with the cluster management service, a second instance will detect the failure. For example, if an instance named `instance1` loses connectivity while executing a scheduled task, the cluster manager notifies the scheduler service that `instance1` is not available. The scheduler service then attempts to clean up any jobs that `instance1` was running at that time. Note that clustered instances claim scheduled tasks in a random order. For more information, refer to Scheduled tasks across a cluster.

Consistency and concurrency across cluster instances is ensured using multi-version concurrency control (MVCC). MVCC provides consistency because each instance updates only the particular revision of the object that was specified in the update.

All instances in a cluster run simultaneously. When a clustered deployment is configured with a load balancer, the deployment works as an active-active high availability cluster. If the database is also clustered, IDM points to the database cluster as a single system.

IDM requires a single, consistent view of all the data it manages, including the user store, roles, schedules, and configuration. If you can guarantee this consistent view, the number and locations of IDM nodes in a cluster will be limited only by your network latency and other network factors that affect performance.

The following diagram shows an IDM deployment where both the IDM instances and the databases are clustered, and accessed through a load balancer:



This chapter describes the changes required to configure multiple IDM instances in a single cluster. It does not include instructions on configuring the various third-party load balancing options.

Important

A clustered deployment relies on system heartbeats to assess the cluster state. For the heartbeat mechanism to work, you *must* synchronize the system clocks of all the machines in the cluster using a time synchronization service that runs regularly. The system clocks must be within one second of each other. For information on how you can achieve this using the Network Time Protocol (NTP) daemon, refer to the NTP RFC.
Note that VM guests do not necessarily keep the same time as the host. You should therefore run a time synchronization service such as NTP on every VM.

IDM cluster configuration

Setting up multiple IDM instances in a cluster involves the following main steps:

1. Ensure that each instance is shut down.
2. Configure each instance to use the same external repository and the same keystore and truststore.
3. Set a unique node ID for each instance.
4. Configure the entire clustered system to use a load balancer or reverse proxy.

To configure an IDM instance as a part of a clustered deployment, follow these steps:

1. Shut down the server if it is running.
2. If you have not already done so, set up a supported repository, as described in Select a repository.

   Each instance in the cluster must be configured to use the same repository; that is, the database connection configuration file (`datasource.jdbc-default.json`) for each instance must point to the same port number and IP address for the database.

   > **Note**
   >
   > The configuration file `datasource.jdbc-default.json` must be the same on all nodes.

   Do not run the data definition language script file in Select a repository for each instance in the cluster—run it just once to set up the tables required for IDM.

   > **Important**
   >
   > If an instance is *not* participating in the cluster, it must *not* share a repository with nodes that are participating in the cluster. Having non-clustered nodes use the same repository as clustered nodes will result in unexpected behavior.

3. Specify a unique node ID (`openidm.node.id`) for each instance, in one of the following ways:

   - Set the value of `openidm.node.id` in the `resolver/boot.properties` file of the instance. For example:

     ```
     openidm.node.id = node1
     ```

   - Set the value in the `OPENIDM_OPTS` environment variable and export that variable before starting the instance. You must include the JVM memory options when you set this variable. For example:

     ```
     export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.node.id=node1" ./startup.sh
     Executing ./startup.sh...
     Using OPENIDM_HOME:   /path/to/openidm
     Using PROJECT_HOME:   /path/to/openidm
     Using OPENIDM_OPTS:   -Xmx1024m -Xms1024m -Dopenidm.node.id=node1
     Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
     Using boot properties at /path/to/openidm/resolver/boot.properties
     -> OpenIDM version "7.3.2"
     OpenIDM ready
     ```

     You can set any value for the `openidm.node.id` , as long as the value is unique within the cluster. The cluster manager detects unavailable instances by their node ID.

     You *must* set a node ID for each instance, otherwise the instance fails to start. The default `resolver/boot.properties` file sets the node ID to `openidm.node.id=node1` .

4. Set the cluster configuration in the `conf/cluster.json` file.

   By default, configuration changes are persisted in the repository so changes that you make in this file apply to all nodes in the cluster.

The default version of the `cluster.json` file assumes that the cluster management service is enabled:

```
{
  "instanceId" : "&{openidm.node.id}",
  "instanceTimeout" : 30000,
  "instanceRecoveryTimeout" : 30000,
  "instanceCheckInInterval" : 5000,
  "instanceCheckInOffset" : 0,
  "enabled" : true
}
```

instanceId

The ID of this node in the cluster. By default, this is set to the value of the instance's `openidm.node.id` that you set in the previous step.

instanceTimeout

The length of time (in milliseconds) that a member of the cluster can be "down" before the cluster manager considers that instance to be in *recovery mode*.

Recovery mode indicates that the `instanceTimeout` of an instance has expired, and that another instance in the cluster has detected that event. The scheduler component of the second instance then moves any incomplete jobs into the queue for the cluster.

instanceRecoveryTimeout

Specifies the time (in milliseconds) that an instance can be in recovery mode before it is considered to be offline.

This property sets a limit after which other members of the cluster stop trying to access an unavailable instance.

instanceCheckInInterval

Specifies the frequency (in milliseconds) that instances check in with the cluster manager to indicate that they are still online.

instanceCheckInOffset

Specifies an offset (in milliseconds) for the check-in timing, when multiple instances in a cluster are started simultaneously.

The check-in offset prevents multiple instances from checking in simultaneously, which would strain the cluster manager resource.

enabled

Specifies whether the cluster management service is enabled when you start the server. This property is set to `true` by default.

> **Important**
>
> If you disable the cluster manager while clustered nodes are running (by setting `"enabled" : false` in an instance's `cluster.json` file), the following happens:
> - The cluster manager thread that causes instances to *check in* is not deactivated.
> - Nodes in the cluster no longer receive cluster *events*, which are used to broadcast configuration changes when they occur over the REST interface.
> - Nodes are unable to detect and attempt to recover failed instances within the cluster.
> - Persisted schedules associated with failed instances cannot be recovered by other nodes.

5. Specify how the instance reads configuration changes. For more information, refer to How IDM Instances Read Configuration Changes.

6. If you are using scheduled tasks, configure persistent schedules so that jobs and tasks are launched only once across the cluster.

7. Configure each node in the cluster to work with host headers. If you're using a load balancer, adjust the default `jetty.xml` configuration, as described in Deploy Securely Behind a Load Balancer

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

8. Make sure that each node in the cluster has the same keystore and truststore. You can do this in one of the following ways:

   - When the first instance has been started, copy the initialized keystore (`/path/to/openidm/security/keystore.jceks`) and truststore (`/path/to/openidm/security/truststore`) to all other instances in the cluster.

   - Use a single keystore that is shared between all the nodes. The shared keystore might be on a mounted filesystem, a Hardware Security Module (HSM) or something similar. If you use this method, set the following properties in the `resolver/boot.properties` file of each instance to point to the shared keystore:

     ```
     openidm.keystore.location=path/to/keystore
     openidm.truststore.location=path/to/truststore
     ```

     - For information on configuring IDM to use an HSM device, refer to Configuring IDM For a Hardware Security Module (HSM) Device.

   - The configuration file `secrets.json` in the `/path/to/openidm/conf` directory must be the same on all the nodes.

9. Start each instance in the cluster.

Important

The audit service logs configuration changes only on the modified instance. Although configuration changes are persisted in the repository, and replicated on other instances by default, those changes are not logged separately for each instance.
Configuration changes are persisted by default, but changes to workflows and scripts, and extensions to the UI are not. Any changes that you make in these areas must be manually copied to each node in the cluster.

Configuration updates in a cluster

IDM can read its configuration from the following locations:

- *Repository.* Each instance reads its configuration from the `configobjects` and `configobjectproperties` tables in a JDBC repository, or from the `ou=config,dc=openidm,dc=forgerock,dc=com` baseDN in a DS repository.
- *Filesystem.* Each instance reads its configuration from the JSON files under its `conf` directory and stores the configuration locally in memory.

  In a clustered deployment, file-based configuration changes must be applied manually across all instances.
- *Memory.* The configuration can diverge if an instance is cut from its cluster due to a networking issue or a misconfigured load balancer. In this case, configuration changes made in the repository might not be detected and the configuration in memory will not be updated.

There are two properties in the `conf/system.properties` file that determine how configuration changes are handled for each instance:

openidm.config.repo.enabled

When this property is set to `true`, the instance reads configuration changes from the repository.

The default setting (`# openidm.config.repo.enabled=false`) indicates that the parameter is true. Uncomment that line to prevent the instance from reading configuration changes from the repository.

openidm.fileinstall.enabled

When this property is set to `true`, the instance reads its configuration from the files in its `conf/` directory.

The default setting (`# openidm.fileinstall.enabled=false`) indicates that the parameter is true. Uncomment that line to prevent the instance from reading file-based configuration changes.

Important

Every node in the cluster must have the identical configuration setting. For example, if you set `openidm.config.repo.enabled=true, openidm.fileinstall.enabled=false` on one node, you must set exactly the same options on every node in the cluster.

Repository-based configuration

Traditional clustered deployments share a *mutable* configuration that is read from a shared repository. The repository initially loads the configuration from the JSON files in the `conf` directory of the first instance that is configured in the cluster. However configuration changes are made, they are written to the repository, and the repository is the authoritative configuration source.

Therefore, a traditional clustered deployment generally has the following configuration:

```
openidm.config.repo.enabled=true
openidm.fileinstall.enabled=false
```

File-based configuration

A file-based configuration lets you store the system configuration in a version-controlled filesystem, and to push a new version out to all nodes when the configuration changes. Using a file-based configuration therefore makes versioning and rolling out new configuration easier than pushing new configuration out over REST.

Container deployments typically require an *immutable* configuration that is read from a file system (such as a Docker image) and stored in memory. The file system is the authoritative configuration source and configuration changes are *not* written to the repository.

Therefore, a container deployment generally has the following configuration:

```
openidm.config.repo.enabled=false
openidm.fileinstall.enabled=true
```

If file-based configuration is used, you *must* ensure that the configuration across instances remains consistent. Because the file-based configuration is not shared between instances, changes made to one node's configuration must be applied manually to all nodes across the cluster.

By default, IDM polls JSON configuration files in each `conf/` directory for changes. For security reasons, it is generally recommended that you disable automatic polling of configuration files to prevent untested configuration changes from disrupting your identity service.

For information on this parameter, refer to Disabling Automatic Configuration Updates.

Scheduled tasks across a cluster

In a clustered environment, the scheduler service looks for pending jobs and handles them as follows:

- Non-persistent (in-memory) jobs execute only on the node that created it.
- Persistent scheduled jobs are picked up and executed by any available node in the cluster that has been configured to execute persistent jobs.
- Jobs that are configured as persistent but *not concurrent* run on only one instance in the cluster at a time. That job will not run again at the scheduled time, on any instance in the cluster, until the current job is complete.

  For example, a reconciliation operation that runs for longer than the time between scheduled intervals will not trigger a duplicate job while it is still running.

In clustered environments, the scheduler service obtains an `instanceID`, and check-in and timeout settings from the cluster management service (defined in the `project-dir/conf/cluster.json` file).

IDM instances in a cluster claim jobs in a random order. If one instance fails, the cluster manager automatically reassigns unstarted jobs that were claimed by that failed instance.

For example, if instance A claims a job but does not start it, and then loses connectivity, instance B can claim that job. If instance A claims a job, starts it, and then loses connectivity, other instances in the cluster cannot claim that job. If the failed instance does not complete the task, the next action depends on the misfire policy, defined in the scheduler configuration.

You can override this behavior with an external load balancer.

If a LiveSync operation leads to multiple changes, a single instance processes all changes related to that operation.

Because all nodes in a cluster read their configuration from a single repository, you must use an instance's `resolver/boot.properties` file to define a specific scheduler configuration for that instance. Settings in the `boot.properties` file are not persisted in the repository, so you can use this file to set different values for a property across different nodes in the cluster.

For example, if your deployment has a four-node cluster and you want only two of those nodes to execute persisted schedules, you can disable persisted schedules in the `boot.properties` files of the remaining two nodes. If you set these values directly in the `scheduler.json` file, the values are persisted to the repository and are therefore applied to all nodes in the cluster.

By default, instances in a cluster are able to execute persistent schedules. The setting in the `boot.properties` file that governs this behavior is:

```
openidm.scheduler.execute.persistent.schedules=true
```

To prevent a specific instance from claiming pending jobs, or processing clustered schedules, set `openidm.scheduler.execute.persistent.schedules=false` in the `boot.properties` file of that instance.

Caution

Changing the value of the `openidm.scheduler.execute.persistent.schedules` property in the `boot.properties` file changes the scheduler that manages scheduled tasks on that node. Because the persistent and in-memory schedulers are managed separately, a situation can arise where two separate schedules have the same schedule name.

For more information about persistent schedules, refer to Persistent schedules.

Manage nodes in a cluster

You can manage clusters and nodes over the REST interface, or using the admin UI.

Manage nodes over REST

You can manage clusters and individual nodes over the REST interface, at the endpoint `openidm/cluster/`. These sample REST commands demonstrate the cluster information that is available over REST:

The following REST request displays the nodes configured in the cluster, and their status.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0"  \
--request GET \
"http://localhost:8080/openidm/cluster?_queryFilter=true"
{
  "result": [
    {
       "_id": "node1",
       "state": "running",
       "instanceId": "node1",
       "startup": "2017-09-16T15:37:04.757Z",
       "shutdown": ""
    },
    {
       "_id": "node2",
       "state": "running",
       "instanceId": "node2",
       "startup": "2017-09-16T15:45:05.652Z",
       "shutdown": ""
    }
  ]
}
```

To check the status of a specific node, include its node ID in the URL. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0"  \
--request GET \
"http://localhost:8080/openidm/cluster/node1"
{
  "_id": "node1",
  "instanceId": "node1",
  "startup": "2017-09-16T15:37:04.757Z",
  "shutdown": "",
  "state": "running"
}
```

Manage Nodes Using the admin UI

The admin UI provides a status widget that lets you monitor the activity and status of all nodes in a cluster.

To add the widget to a Dashboard, click Add Widget then scroll down to System Status > Cluster Node Status and click Add.

The cluster node status widget shows the current status and number of running jobs of each node.

Select Status to obtain more information on the latest startup and shutdown times of that node. Select Jobs to obtain detailed information on the tasks that the node is running.

The widget can be managed in the same way as any other dashboard widget. For more information, refer to Manage Dashboards.

Remove Nodes

IDM automatically addresses cluster node removal. Shut down the instance to remove, and IDM no longer utilizes the node.

Considerations when removing an existing node from the cluster:

- Ensure a clustered reconciliation is not taking place when removing the node as this could cause unexpected behavior.
- Remnants of the node will still exist in the chosen IDM database. For example, you may refer to the removed node still appearing in the cluster node status widget in the Dashboard (for more context on this, refer to Manage nodes using the admin UI).

Example using DS

1. Shut down all instances of IDM.
2. Delete the old instance record from DS:
    1. Locate the DN of the node using the `ldapsearch` command:

```
./ldapsearch -D "uid=admin" -w password -h localhost -p 1389 -b
"dc=openidm,dc=forgerock,dc=com" "uid=node1"

dn: uid=node1,ou=states,ou=cluster,dc=openidm,dc=forgerock,dc=com
objectClass: uidObject
objectClass: fr-idm-generic-obj
objectClass: top
fr-idm-json: {"recoveryAttempts":
1,"detectedDown":"0000001660588116038","type":"state","recoveryFinished":"0000001660588116096","inst
3,"recoveringTimestamp":"0000001660588116038","recoveryStarted":"0000001660588116038","shutdown":"00
uid: node1
```

> **Note**
>
> The name of the node instance can be found in the `openidm/resolver/boot.properties` file by the value of the `openidm.node.id` attribute.

    2. Delete the node record:

```
./ldapdelete -D "uid=admin" -w password -h localhost -p 1389 -b
"dc=openidm,dc=forgerock,dc=com"
"uid=node1,ou=states,ou=cluster,dc=openidm,dc=forgerock,dc=com"

# DELETE operation successful for DN
uid=node1,ou=states,ou=cluster,dc=openidm,dc=forgerock,dc=com
```

3. Start all operational IDM nodes.

Example using Oracle database

1. Shut down all instances of IDM.
2. Delete references to the old instance from `clusterobjects` and the associated rows from `clusterobjectproperties` in the database:
    1. Locate the name of the node, in this case `node1`:

```
SELECT * FROM clusterobjects;

|-------------------------------------------------|
| id | objecttypes_id | objectid | rev   | fullobject|
|----|----------------|----------|-------|-----------|
|  1 |              2 | node1    | 43789 | {"redacted"}
|----|----------------|----------|-------|-----------|
```

    2. Delete the references of the old node from the `clusterobjects` table:

```
DELETE FROM openidm.clusterobjects WHERE objectid = 'node1';

|------------------|----------------------|-------------------|---------------------|
| clusterobjects_id | propkey              | proptype          | propvalue           |
|------------------|----------------------|-------------------|---------------------|
|                1 | /recoveryAttempts    | java.lang.Integer | 0                   |
|                1 | /_rev                | java.lang.String  | 43790               |
|                1 | /detectedDown        | java.lang.String  | 0000000000000000000 |
|------------------|----------------------|-------------------|---------------------|
```

    3. Delete references of the old node from the `clusterobjectproperties` table:

```
DELETE FROM openidm.clusterobjectproperties WHERE clusterobjects_id = 'node1';
```

3. Start all operational IDM nodes.

Host and port information

To change the default IDM hostname or listening ports, edit the applicable entry in `openidm/resolver/boot.properties`:

```
openidm.port.http=8080
openidm.port.https=8443
openidm.port.mutualauth=8444
openidm.host=localhost

openidm.auth.clientauthonlyports=8444
```

8080

HTTP access to the REST API, requiring IDM 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 IDM authentication

8444

HTTPS access to the REST API, requiring SSL mutual authentication. Clients that present certificates found in the truststore (`openidm/security/`) are granted access to the system.

Property files

This section lists the `*.properties` files used to configure IDM. Apart from the `boot.properties` file, these files are located in your project's `conf/` directory. This section does not include the `*.properties` files associated with OpenICF connectors.

> **Important**
>
> After making changes to any `*.properties` file, you must restart IDM for the changes to take effect.

`boot.properties`

The `boot.properties` file is the property resolver file used for property substitution, and it is located in the `/path/to/openidm/resolver` directory. This file lets you set variables used in other configuration files, including config.properties and system.properties.

`config.properties`

The `config.properties` file is used for two purposes:

- To set OSGi bundle properties.
- To set Apache Felix properties.

For more information about each item in `config.properties`, refer to Apache Felix Framework Configuration Properties.

logging.properties

The `logging.properties` file configures JDK logging for IDM.

system.properties

The `system.properties` file is used to bootstrap java system properties such as:

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

- Jetty log settings, based on the Jetty container bundled with IDM. IDM bundles Jetty version 12.0.19.
- Configuration Changes
- Quartz updates, as described in Quartz Best Practices documentation.

  `org.terracotta.quartz`
- A common transaction ID, as described in Configure the audit service.

Embedded Jetty configuration

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

IDM includes an embedded Jetty web server.

To configure the embedded Jetty server, edit `openidm/conf/jetty.xml`. IDM delegates most of the 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 options.

The Jetty configuration can reference configuration properties (such as port numbers and keystore details) from your `resolver/boot.properties` file.

IDM configuration properties in Jetty

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

IDM exposes a `Param` class that you can use in `jetty.xml` to include IDM-specific configuration. The `Param` class exposes Bean properties for common Jetty settings and generic property access for other arbitrary settings.

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 refer to the bundled `jetty.xml` for further examples.

The following explicit Bean properties are available; they map either to the `boot.properties` in the `openidm/resolver/` subdirectory, or the `secrets.json` file in your project's `conf/` subdirectory.

port

Maps to `openidm.port.http`

port

Maps to `openidm.port.https`

port

Maps to `openidm.port.mutualauth`

keystoreType

Maps to `mainKeyStore `storeType`

keystoreProvider

Maps to `mainKeyStore `providerName`

keystoreLocation

Maps to `mainKeyStore `file`

keystorePassword

Maps to `mainKeyStore `storePassword`

truststoreLocation

Maps to `mainTrustStore `file`

truststorePassword

Maps to `mainTrustStore `storePassword`

Generic Properties

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

Jetty default settings

By default, the embedded Jetty server uses the following settings:

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

- The HTTP, SSL, and Mutual Authentication ports defined in IDM.
- The same keystore and truststore settings as IDM.
- 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.

Additional servlet filters

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

You can register generic servlet filters in the embedded Jetty server to perform additional filtering tasks on requests to or responses from IDM. For example, you might want to use a servlet filter to protect access to IDM with an access management product. Servlet filters are configured in files named `openidm/conf/servletfilter-name.json`. These servlet filter configuration files define the filter class, required libraries, and other settings.

A sample servlet filter configuration is provided in the `servletfilter-cors.json` file in the `/path/to/openidm/conf` directory.

The sample servlet filter configuration file is shown below:

```
{
    "classPathURLs" : [ ],
    "systemProperties" : { },
    "requestAttributes" : { },
    "scriptExtensions" : { }.
    "initParams" : {
       "allowedOrigins" : "https://localhost:&{openidm.port.https}",
       "allowedMethods" : "GET,POST,PUT,DELETE,PATCH",
       "allowedHeaders" : "accept,x-openidm-password,x-openidm-nosession,
                           x-openidm-username,content-type,origin,
                           x-requested-with",
       "allowCredentials" : true,
       "chainPreflight" : false
    },
    "urlPatterns" : [
       "/*"
    ],
    "filterClass" : "org.eclipse.jetty.ee10.servlets.CrossOriginFilter"
}
```

The sample configuration includes the following properties:

`classPathURLs`

The URLs to any required classes or libraries that should be added to the classpath used by the servlet filter class.

`systemProperties`

Any additional Java system properties required by the filter.

`requestAttributes`

The HTTP Servlet request attributes that will be set when the filter is invoked. IDM expects certain request attributes to be set by any module that protects access to it, so this helps in setting these expected settings.

`scriptExtensions`

Optional script extensions to IDM. Currently only `augmentSecurityContext` is supported. A script that is defined in `augmentSecurityContext` is executed after a successful authentication request. The script helps to populate the expected security context. For example, the login module (servlet filter) might select to supply only the authenticated user name, while the associated roles and user ID can be augmented by the script.

Supported script types include `"text/javascript"` and `"groovy"`. The script can be provided inline (`"source":script source`) or in a file (`"file":filename`). The sample filter extends the filter interface with the functionality in the script `script/security/populateContext.js`.

`filterClass`

The servlet filter that is being registered.

The following additional properties can be configured for the filter:

`httpContextId`

The HTTP context under which the filter should be registered. The default is `"openidm"`.

`servletNames`

A list of servlet names to which the filter should apply. The default is `"OpenIDM REST"`.

`urlPatterns`

A list of URL patterns to which the filter applies. The default is `["/*"]`.

`initParams`

Filter configuration initialization parameters that are passed to the servlet filter `init` method. For more information, refer to http://docs.oracle.com/javaee/5/api/javax/servlet/FilterConfig.html.

Secure protocol configuration

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

The Jetty configuration for inbound connections to IDM supports a number of protocols and cipher suites.

Enabled *protocols* are explicitly listed in the `includeProtocols` list in the `conf/jetty.xml` file. Only `TLSv1.2` and `TLSv1.3` are enabled by default:

```
...
<Array  id= "includedProtocols" type="java.lang.String">
    <!-- Only support TLS v1.2 and v1.3 -->
    <Item>TLSv1.2</Item>
    <Item>TLSv1.3</Item>
</Array>
...
```

To disable a particular protocol, remove it from the `includedProtocols` list. To add support for a weaker protocol, add it to the list. For example:

```
...
<Array  id= "includedProtocols" type="java.lang.String">
    <Item>TLSv1.2</Item>
    <Item>TLSv1.3</Item>
    <Item>SSLv3.0</Item>
</Array>
...
```

Important

It is highly recommended that you do not enable weaker protocols such as SSL, and TLS versions prior to 1.2. These protocols use outdated algorithms and are generally considered insecure.

Enabled *cipher suites* for each protocol are listed in the `includedCipherSuites` list in `conf/jetty.xml`:

```
...
<Array id="includedCipherSuites" type="java.lang.String">
   <!-- TLS 1.3 cipher suites -->
   <Item>TLS_AES_128_GCM_SHA256</Item>
   <Item>TLS_AES_256_GCM_SHA384</Item>
   <Item>TLS_CHACHA20_POLY1305_SHA256</Item>

   <!-- TLS 1.2 cipher suites -->
   <Item>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</Item>
   <Item>TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256</Item>
   <Item>TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256</Item>
   <Item>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384</Item>
   <Item>TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256</Item>
   <Item>TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256</Item>
   <Item>TLS_DHE_RSA_WITH_AES_256_GCM_SHA384</Item>
   <Item>TLS_DHE_RSA_WITH_AES_128_GCM_SHA256</Item>
</Array>
...
```

To add support for additional cipher suites, add them as `<Item>`s in this list.

Jetty thread settings

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

To change the Jetty thread pool settings, add the following excerpt to your project's `conf/config.properties` file:

```
# Jetty maxThreads (default 200)
org.ops4j.pax.web.server.maxThreads=${org.ops4j.pax.web.server.maxThreads}
# Jetty minThreads (default 8)
org.ops4j.pax.web.server.minThreads=${org.ops4j.pax.web.server.minThreads}
# Jetty idle-thread timeout milliseconds (default 60000)
org.ops4j.pax.web.server.idleTimeout=${org.ops4j.pax.web.server.idleTimeout}
```

To override these defaults, set a corresponding `OPENIDM_OPTS` variable when you start IDM. For example:

```
export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dorg.ops4j.pax.web.server.maxThreads=768" /path/to/openidm/
startup.sh
Executing ./startup.sh...
Using OPENIDM_HOME:   /path/to/openidm
Using PROJECT_HOME:   /path/to/openidm
Using OPENIDM_OPTS:   -Xmx1024m -Xms1024m -Dorg.ops4j.pax.web.server.maxThreads=768
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
-> OpenIDM version "8.0.0" (revision: unknown)
OpenIDM ready
```

> **Important**
>
> You cannot use property substitution to set these properties.
> You cannot adjust Jetty's thread settings in the `conf/jetty.xml` file. If you uncomment the excerpt of `jetty.xml` that starts with `<!--<Arg name="threadpool">...`, errors display in the IDM log.

Gzip compression for HTTP responses

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

IDM uses the Jetty Gzip handler to compress HTTP responses. The default Gzip handler configuration, in `conf/jetty.xml`, is as follows:

> **Important**
>
> In Jetty 12, the `compressionLevel` and `excludedAgentPatterns` properties have been removed from the Gzip handler. Learn more in Discontinued functionality.

```
...
<Call name="insertHandler">
    <Arg>
        <New id="GzipHandler" class="org.eclipse.jetty.server.handler.gzip.GzipHandler">
            <Set name="minGzipSize"><Property name="jetty.gzip.minGzipSize" default="2048"/></Set>
            <Set name="inflateBufferSize"><Property name="jetty.gzip.inflateBufferSize" default="0"/></Set>
            <Set name="syncFlush"><Property name="jetty.gzip.syncFlush" default="false" /></Set>
            <Set name="includedMethodList"><Property name="jetty.gzip.includedMethodList" default="GET" /></Set>
            <Set name="excludedMethodList"><Property name="jetty.gzip.excludedMethodList" default="" /></Set>
        </New>
    </Arg>
</Call>
...
```

Adjust this configuration if the default does not suit your deployment. Configuration properties are as follows:

`minGzipSize`

Content is compressed only if the content length is unknown or is greater than the `minGzipSize`. By default, content is compressed only if the response is greater than 2048MB.

`inflateBufferSize`

Number of bytes in the request decompression buffer. The default setting is `-1`, which disables this feature. Use this feature only if you want to compress large POST/PUT request payloads. Be aware that this setting exposes a potential Zip bomb risk.

`syncFlush`

By default, this setting is `false`, which lets the deflater determine how much data to accumulate, before it produces output. This achieves the best compression. When `true`, this setting forces flushing of the buffer of data to compress. This can result in poor compression.

`includedMethodList`

A list of HTTP methods to compress. By default, only GET requests are compressed.

`excludedMethodList`

A list of HTTP methods that should not be compressed.

Upgrade

This guide shows you how to upgrade an existing deployment to the latest ForgeRock® Identity Management release.

Quick Start

Migrate Configuration

Migrate an existing IDM configuration to IDM 7.3.

Update Repository

Update an existing repository or install a new repository for IDM 7.3.

Migrate Data

Move the data in an existing IDM repository to an updated deployment.

The upgrade process is largely dependent on your deployment and on the extent to which you have customized IDM. Engage ForgeRock Support Services for help in upgrading an existing deployment. Also, read the Release notes before you start an upgrade; specifically, Incompatible changes.

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

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

About upgrades

The automated update process available with previous IDM versions is no longer supported. This chapter describes the manual process required to upgrade an existing IDM deployment. At a high level, the manual update process involves the following steps:

1. Install IDM 7.3.

2. Migrate your existing IDM configuration to the new installation.
3. Update your repository.
4. Test your scripts and customizations work as expected.
5. Migrate existing data to the new installation.

Supported upgrade paths

The following table contains information about the supported upgrade paths to IDM 7.3:

Upgrade Paths

Version	Upgrade Supported to IDM 7.3
IDM 7.2.x	YES
IDM 7.1.x	YES
IDM 7.0.x	YES
IDM 6.5.x	YES
IDM 6.0.x	YES
IDM 5.5.x	YES
IDM 5.0.x	YES

Important

Depending on how you have customized your deployment, there might be incompatible configuration changes when you upgrade from versions prior to IDM 6.5.x. Read the upgrade documentation for each interim release and apply all required script and configuration changes.

Before you upgrade

Fulfill these requirements before you upgrade IDM, especially before upgrading the software in a production environment. Also refer to the requirements listed in Before you install and the changes listed in Incompatible changes.

Before you start, verify that you have a supported Java version installed:

Supported Java Versions

Vendor	Versions
OpenJDK, including OpenJDK-based distributions: • AdoptOpenJDK/Eclipse Temurin • Amazon Corretto • Azul Zulu • Red Hat OpenJDK **Note** ForgeRock tests most extensively with AdoptOpenJDK/Eclipse Temurin. ForgeRock recommends using the HotSpot JVM.	17*
Oracle Java	17*

* Version 17.0.9 or higher.

If the server uses an older version that is no longer supported, install a newer Java version before you update, and follow the instructions in Java requirements.

Then, follow these steps:

1. Back up your existing deployment by archiving the `openidm` directory and creating a backup of the repository and all other applicable databases.

   Note

   If you use workflow, you must manually dump the workflow database tables, and then import them *before* you start the new instance of IDM for the first time. The workflow database tables start with the prefix `ACT_`. For information on how to dump/import individual tables, refer to the documentation for your database.

2. To save a record of the audit logs from your existing IDM installation, manually copy the log files from the `/path/to/openidm/audit/` directory, before you start the upgrade.

3. Download and extract `IDM-7.3.2.zip` from the Backstage download site.

Migrate your configuration

This chapter covers the steps required to migrate your IDM configuration to IDM 7.3.

There is no automated way to migrate a customized configuration to IDM 7.3, so you must migrate customized configuration files manually. Assuming you are upgrading from IDM 7.2.x, there are three ways to do this:

- Use the new IDM 7.3 configuration files as a base, and copy any customizations you have made to the new files.

  This is the preferred option, particularly if you have used version control on your configuration and can determine the exact changes you have applied.

- Use your existing configuration files as a base, and add any new IDM 7.3 configuration to your existing files.

- Use your existing configuration "as is" with no IDM 7.3 changes.

In most cases, a customized IDM 7.2.x configuration will work without further modification on IDM 7.3.

Migrate configuration files

For customized files in your project's `conf/` directory, check that the customizations are compatible with the changes outlined in Incompatible changes. If there are no incompatible changes, either copy your old configuration files to your IDM 7.3 installation, or copy any customization into the corresponding new configuration files.

> **Note**
>
> If you create custom configuration files, ForgeRock recommends not using spaces or special characters in the filenames, in accordance with the OSGi specification.

Migrate `boot.properties`

On the IDM 7.3 installation, edit the `resolver/boot.properties` file to match any customizations that you made on your IDM 7.2.x server. Specifically, check the following elements:

- The HTTP, HTTPS, and mutual authentication ports.

  If you changed the default ports in your IDM 7.2.x deployment, make those same changes in the new `boot.properties` file.

- Check that the keystore and truststore passwords match the current passwords for the keystore and truststore of your existing IDM deployment.

Migrate security settings

Copy the contents of your IDM 7.2.x `security/` folder to the IDM 7.3 installation.

> **Warning**
>
> If you do not copy your old truststore and keystore files to your new instance, you will be unable to decrypt anything that was encrypted by your old instance of IDM.

Migrate custom scripts

Migrate any custom scripts or default scripts *that you have modified* to the `script` directory of your IDM 7.3 instance. In general, custom and customized scripts should be located in the `openidm/script` directory of your existing IDM deployment.

For custom scripts, review Incompatible changes. If you are confident that the scripts will work as intended on IDM 7.3, copy these scripts to the new instance.

If you modified a default IDM script, compare the default versions of the IDM 7.2.x and IDM 7.3 scripts. If nothing has changed between the default versions, review your customizations against Incompatible changes. If a default script has changed since the IDM 7.2.x release, test that your customizations work with the new default script. If you are confident that your changes will work as intended on the new version, copy the customized scripts to the new `script` directory.

> **Note**
>
> If you modify any shell scripts, such as `startup.sh`, you must migrate your changes manually to the new version of the script.

Migrate custom bundles

If your existing deployment includes any custom JAR files in the `bundles` directory, migrate these to the new deployment. Pay particular attention to any files that support JDBC database drivers.

Migrate provisioner files

Change any customized provisioner configurations in your existing deployment to point to the connectors that are provided with IDM 7.3. Specifically, make sure that the `connectorRef` properties reflect the new connector versions, where applicable. For example:

```
"connectorRef" : {
    "bundleName": "org.forgerock.openicf.connectors.ldap-connector",
    "bundleVersion": "[1.4.0.0,1.6.0.0)",
    "connectorName": "org.identityconnectors.ldap.LdapConnector"
},
```

Alternatively, copy the connector .jar files from your existing deployment into the `openidm/connectors` directory of the new installation.

Migrate UI customizations

If you have customized the admin UI, review any custom UI files from your IDM 7.2.x deployment (generally in the `openidm/ui/admin/extension` directory), and compare them against the corresponding IDM 7.3 files.

For each customized file, copy the corresponding default IDM 7.3 UI files to a `openidm/ui/admin/extension` directory on the new instance.

Apply your customizations to files in the new `openidm/ui/admin/extension` directory.

Update the repository

When you have migrated your configuration to the new IDM installation, you need to handle the data that is stored in your repository. There are two options to update a repository:

- Upgrade your existing IDM 7.x repository.
- Create a new IDM 7.3 repository, then migrate your data to the new repository.

When you have upgraded the repository, or created a new repository, start the IDM server and test that all your scripts are working as expected, before migrating your data.

Upgrade an existing repository

Upgrading an existing repository means that you do not need to migrate data. However, you must run a series of scripts that modify the repository, to use the new features in IDM 7.3.

> **Important**
>
> Because the repository upgrade scripts are incremental, you must review each major version upgrade after your current release. For example, when upgrading from 6.5.x to 7.3.x, review the upgrade process and scripts for 7.0.x, 7.1.x, 7.2.x, and 7.3.x (this version).
> Repository upgrade procedures:
>
> - Upgrade an existing repository (7.0.x)
> - Upgrade an existing repository (7.1.x)
> - Upgrade an existing repository (7.2.x)

Prepare an existing repository for IDM 7.3 as follows:

1. Shut down IDM, if it is running.
2. Clear all `configobjects` related tables. For example, in MySQL run:

```
DELETE FROM openidm.configobjects;
DELETE FROM openidm.configobjectproperties;
```

3. From your IDM 8.0 installation, run the schema update scripts for your database type:
   - For Microsoft SQL (MSSQL) only, `/path/to/openidm/db/mssql/scripts/updates/00-update-to-nvarchar.sql`. This script updates the deprecated SQL data type `NTEXT` to `NVARCHAR(MAX)`.

     > **Important**
     >
     > This script is only required for explicit `managed_user` tables.

4. If you are using workflow, you must run the Flowable upgrade scripts for your database type. These upgrade scripts are incremental and must be run in order, starting with the correct script based on your current Flowable version.
   1. To determine your current Flowable version, check the `/path/to/openidm/bundle/flowable-engine-versionNumber.jar` file in your old IDM installation.
   2. Run the upgrade scripts from `/path/to/openidm/db/database-type/scripts/updates/` in order, starting with your current flowable version:
      1. `flowable.database-type.upgradestep.6.6.0.to.6.7.0.all.sql`
      2. `flowable.database-type.upgradestep.6.7.0.to.6.7.1.all.sql`
      3. `flowable.database-type.upgradestep.6.7.1.to.6.7.2.all.sql`
      4. `flowable.database-type.upgradestep.6.7.2.to.6.8.0.all.sql`
5. Launch IDM and run the following Groovy script to clear the `reconprogressstate` data in your repository:

```
def result = openidm.query(
  "repo/reconprogressstate", [ "_queryFilter" : "true", "_fields" : "_id" ]).result;
for ( item in result ) {
  openidm.delete("repo/reconprogressstate/" + item["_id"], null);
}
return result.size() + " reconprogressstate records deleted";
```

This script works for all repository types and can be sent as a REST call. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "type":"groovy",
  "source":"def result = openidm.query(\"repo/reconprogressstate\", [ \"_queryFilter\" : \"true\",
\"_fields\" : \"_id\" ]).result; for ( item in result ) { openidm.delete(\"repo/reconprogressstate/
\" + item[\"_id\"], null); }; return result.size() + \" reconprogressstate records deleted\";"
}' \
"http://localhost:8080/openidm/script?_action=eval"
"1 reconprogressstate records deleted"
```

6. Verify that all scripts and functions behave as expected.

Create a new repository

1. Set up a new repository, following the steps in Select a repository. A new repository is already configured for all the new capabilities in IDM, but does require migrating existing data to that repository.

   If you create a new repository, you must still update your configuration files to use the new features.

2. After you have set up the new repository, migrate your data to that repository.

Migrate data

The data migration service helps you move information stored in an IDM repository to a new deployment. You can use this service when you are upgrading to a new version, or when you are migrating to a different repository type. The migration service is off by default. To enable it, copy `migration.json` from `samples/example-configurations/conf/` into your `conf/` directory, and set `"enabled": true`.

Migration is run from your new installation through IDM's recon service, using your previous deployment as a data source. The data migration service supports importing information from IDM instances back to version 4. If you are migrating from a version of IDM earlier than that, you will need to follow previous update instructions to get your deployment into a state where it can be migrated using this service.

> **Note**
>
> Because the migration service migrates information that may be encrypted, such as passwords, you must make sure you have copied the `truststore` and `keystore` files from your previous deployment *before* you start the migration.

- Internal Roles
- Internal Users
- Internal User Metadata
- Managed Roles
- Managed Users
- Managed Assignments
- Links and Relationships
- Scheduler jobs

> **Note**
>
> If you are migrating scheduler jobs from IDM 4.0 or 4.5, you will need to modify the entry in `migration.json` to be:
>
> ```
> {
>     "source" : "scheduler",
>     "target" : "scheduler/job"
> }
> ```

If you have additional object types (for example, managed devices), modify `migration.json` to include these objects.

Configure the Migration Service

The data migration service is configured through `migration.json`. The default file assumes a default schema; modify the file if you have added custom managed data. The `migration.json` file can have the following properties:

enabled

Boolean, `true` or `false`. Enables the migration service.

connection

Configures the connection to the source IDM instance you are migrating from. Available properties:

instanceUrl

The URI for the source IDM instance.

authType

The authentication mechanism to the source IDM instance. Can be `basic` (username/password) or `bearer` (authentication using AM bearer tokens).

userName

Used for authenticating to the source IDM instance, if the `authType` is `basic`.

password

Used for authenticating to the source IDM instance, if the `authType` is `basic`.

clientId

Used for authenticating to the source IDM instance, if the `authType` is `bearer`.

clientSecret

Used for authenticating to the source IDM instance, if the `authType` is `bearer`.

tokenEndpoint

Used for authenticating to the source IDM instance, if the `authType` is `bearer`.

scope (optional)

List of OAuth scopes.

scopeDelimiter (optional)

Delimiter for the list of OAuth scopes.

tlsVersion (optional)

Lets you override the default TLS version.

connectionTimeout (optional)

Timeout for connecting to the source IDM instance (defaults to `10s`).

reuseConnections (optional)

Lets you override the default setting (defaults to `true`).

retryRequests (optional)

Lets you override the default setting (defaults to `true`).

hostnameVerifier (optional)

The SSL hostname verification policy. Specifies whether the host name presented by the remote server certificate is verified upon establishing new SSL connections (defaults to `STRICT`). Possible values:

- `STRICT` : Requires that the host name match the host name presented in the certificate. Wild-cards only match a single domain.
- `ALLOW_ALL` : Accepts any host name (disables host name verification).

maxConnections (optional)

Lets you override the default maximum number of connections (default is `64`).

proxy (optional)

Lets you specify connection through a proxy server. Includes the following properties:

`proxyUri`

The proxy host and port to which IDM should connect.

`userName`

The user account to connect to the remote proxy.

`password`

The password of the proxy user.

socketTimeout

The TCP socket timeout, when waiting for HTTP responses. If you do not set a duration, the default is no timeout.

Example valid duration values:

- 4 days
- 59 minutes and 1 millisecond
- 1 minute and 10 seconds
- 42 millis
- unlimited
- none
- zero

mappings

A list of the endpoints that will be migrated from your old IDM instance to your new instance, expressed as mappings between the old and new instances. The complete list of mapping properties is the same as any regular synchronization mapping. Properties with particular significance for data migration include the following:

allowEmptySourceSet

Specifies whether the migration service should continue if it encounters an empty source mapping. This is enabled by default.

correlationQuery

You can specify a custom correlation query. By default, this is:

```
"var map = {'_queryFilter': '_id eq \"' + source._id + '\"'}; map;"
```

For more information about writing correlation queries, refer to Correlate source objects with existing target objects.

enableLinking

Specifies whether links are maintained between source and target objects. If `enableLinking` is set to `false`, links are not maintained. This is the default behavior for the migration service, where it is expected that you will run the migration only once. If you intend to run the migration more than once, set this parameter to `true`.

onCreate

The script used by the migration service for creating the data that is being migrated to the new installation. By default, this points to a Groovy script: `update/mapLegacyObject.groovy`.

onUpdate

The script used by the migration service for updating the data that is being migrated in the new installation. By default, this points to a Groovy script: `update/mapLegacyObject.groovy`.

policies

An array of policies to apply to the data being migrated.

properties

An array of properties to perform additional actions on, such as modifying the contents of a property during the migration. This follows the pattern you would find in a standard reconciliation. For more information about transforming data during a reconciliation, refer to Transform Attributes in a Mapping.

reconSourceQueryPageSize

Specifies the number of results to return per page, if paging is turned on. By default, 1000 results per page are returned.

reconSourceQueryPaging

Specifies whether the migration service should use paging when querying the source IDM instance. By default, this is set to `false`. Turn paging on if you have a large data set and are concerned about memory usage.

For large data sets, you might be able to improve migration performance by turning paging on and increasing the query page size (using `reconSourceQueryPageSize`). The most effective page size will vary, depending on the available resources.

runTargetPhase

Specifies whether the migration should run the target phase of reconciliation. By default, this is set to `false` as there is no data in the target repository.

source

This is the only property that is *required* for data migration. The source should be the path to the resource within the repo; for example, `repo/managed/user`.

> **Important**
>
> By default, the migration services use the `repo` endpoint, rather than the `managed` endpoint for both the `source` and the `target`. Create, read, update, and delete operations will therefore not trigger an implicit synchronization to the target resource.

sourceQuery

The query on the source system, used to find all objects to be migrated. Defaults to `"_queryFilter" : "true&fields=_id"`, which returns the IDs of all source objects.

You can improve migration performance by returning the whole source entry (setting the `sourceQuery` to `"_queryFilter" : "true"`).

> **Important**
>
> If you are migrating from IDM 6.5.x
> Any explicitly mapped resource coming from `repo/<mappingName>` must include:
>
> ```
> "sourceQuery": {
>     "_queryFilter": "true",
>     "_fields": ""
> }
> ```

sourceQueryFullEntry

(Optional). Specifies whether the defined source query returns full object data (`true`) or IDs only (`false`). Defaults to `true`.

If you do not set this parameter, IDM attempts to detect whether the full object is returned, based on the query results.

target

The path to the resource within the target repository. By default, this will be the same as the source path.

validSource

You can specify a script to validate the source object prior to migration. By default, this property is empty.

endpoint

By default, the migration service endpoint is `migration`. You can use the `endpoint` property to change this if needed.

> **Note**
>
> Because the data migration service performs a reconciliation between your old installation and your new installation, the general reconciliation optimizations also apply to the data migration service. For more information about reconciliation optimization, refer to Tuning reconciliation performance.

Run the Data Migration

Before you run your migration, make sure that you have done the following:

- Paused any scheduled jobs on the source deployment.
- Configured your `conf/migration.json` and `update/mapLegacyObject.groovy` files on the new IDM installation.
- Moved your configuration files from the old deployment to the new one.
- If you use workflow, you must manually dump the workflow database tables, and then import them *before* you start the new instance of IDM for the first time. The workflow database tables start with the prefix `ACT_`. For information on how to dump/import individual tables, refer to the documentation for your database.

When you launch the new IDM installation, a new `migration` endpoint should be available. This endpoint supports the following actions:

- `migrate`: Triggers a migration of all legacy objects from the remote system. Optionally takes a `mapping` parameter in order to specify a specific mapping to migrate. For example:

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

- `status`: Returns the last status for all reconciliations triggered by the migration service.
- `mappingConfigurations`: Returns the full list of migration mapping configurations.
- `mappingNames`: Returns the list of migration mapping names.

The period of time a migration takes will depend on the amount of information being migrated. Migrated data will retain the same object IDs they had in the previous deployment.

> **Note**
>
> If requests sent to the source server include an `X-Requested-With` header, the value of the header will be set to `RemoteIDMProxy`.

Upgrade a clustered deployment

Follow these general steps when you are updating servers in a cluster:

1. Redirect client traffic to a different IDM system or cluster.
2. Shut down every node in the cluster.
3. Update one node in the cluster.
4. Clone the first node to the other nodes in that cluster.

Update to a maintenance release

The Maintenance releases incorporate a collection of fixes and minor RFEs. IDM 7.3.2 is the latest maintenance release for IDM 7.3. To upgrade an existing IDM 7.3.x deployment, follow these steps:

1. Download and extract the IDM 7.3.2 binary from the Backstage download site.
2. Copy any customized configuration files, scripts, or workflow definitions from your existing deployment to the comparable directory in your 7.3.2 deployment.
3. If you're still running an earlier version of IDM 7.3.x, copy the `conf/authentication.json` file from your existing deployment to the `conf/` directory in your 7.3.2 deployment.
4. Copy the keystore and truststore from your existing deployment to the 7.3.2 deployment. For example:

   ```
   cp -r /path/to/openidm73x/security /path/to/openidm732
   ```

5. Configure the IDM 7.3.2 server to point to your existing repository:
   - If you're using an external DS repository, verify the accuracy of the `conf/repo.ds.json` file in your new deployment.
   - If you're using a JDBC repository, verify the accuracy of the following files in your new deployment:
     - `conf/repo.jdbc.json`
     - `conf/datasource.jdbc-default.json`
     - `resolve/boot.properties` (particularly the values for `openidm.repo.host` and `openidm.repo.port`)
6. If you're using workflow, you must run the Flowable upgrade scripts for your database type. These upgrade scripts are incremental and must be run in order, starting with the correct script based on your current Flowable version.
   1. To determine your current Flowable version, check the `/path/to/openidm/bundle/flowable-engine-versionNumber.jar` file in your old IDM installation.
   2. Run the upgrade scripts from `/path/to/openidm/db/database-type/scripts/updates/` in order, starting with your current flowable version:
      1. `flowable.database-type.upgradestep.6.6.0.to.6.7.0.all.sql`

2. `flowable.database-type.upgradestep.6.7.0.to.6.7.1.all.sql`
3. `flowable.database-type.upgradestep.6.7.1.to.6.7.2.all.sql`
4. `flowable.database-type.upgradestep.6.7.2.to.6.8.0.all.sql`

7. Shut down your existing IDM 7.3.x server.
8. Start your IDM 7.3.2 server.

Setup

In this guide, you will learn about the core ForgeRock Identity Management (IDM) IDM architecture, the IDM configuration model, and how to get a basic IDM deployment up and running after installation.

Architecture

IDM architecture, component modules, and services.

Configuration

IDM configuration.

Command-Line Interface

IDM command-line interface (CLI) and utilities.

User Interface

IDM's user interfaces.

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

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

Architectural overview

This topic introduces the IDM architecture, and describes component modules and services, such as:

- How IDM uses the OSGi framework as a basis for its modular architecture.
- How the infrastructure modules provide the features required for IDM's core services.
- What those core services are and how they fit in to the overall architecture.
- How IDM provides access to the resources it manages.

Modular framework

IDM implements infrastructure modules that run in an OSGi framework. It exposes core services through RESTful APIs to client applications.

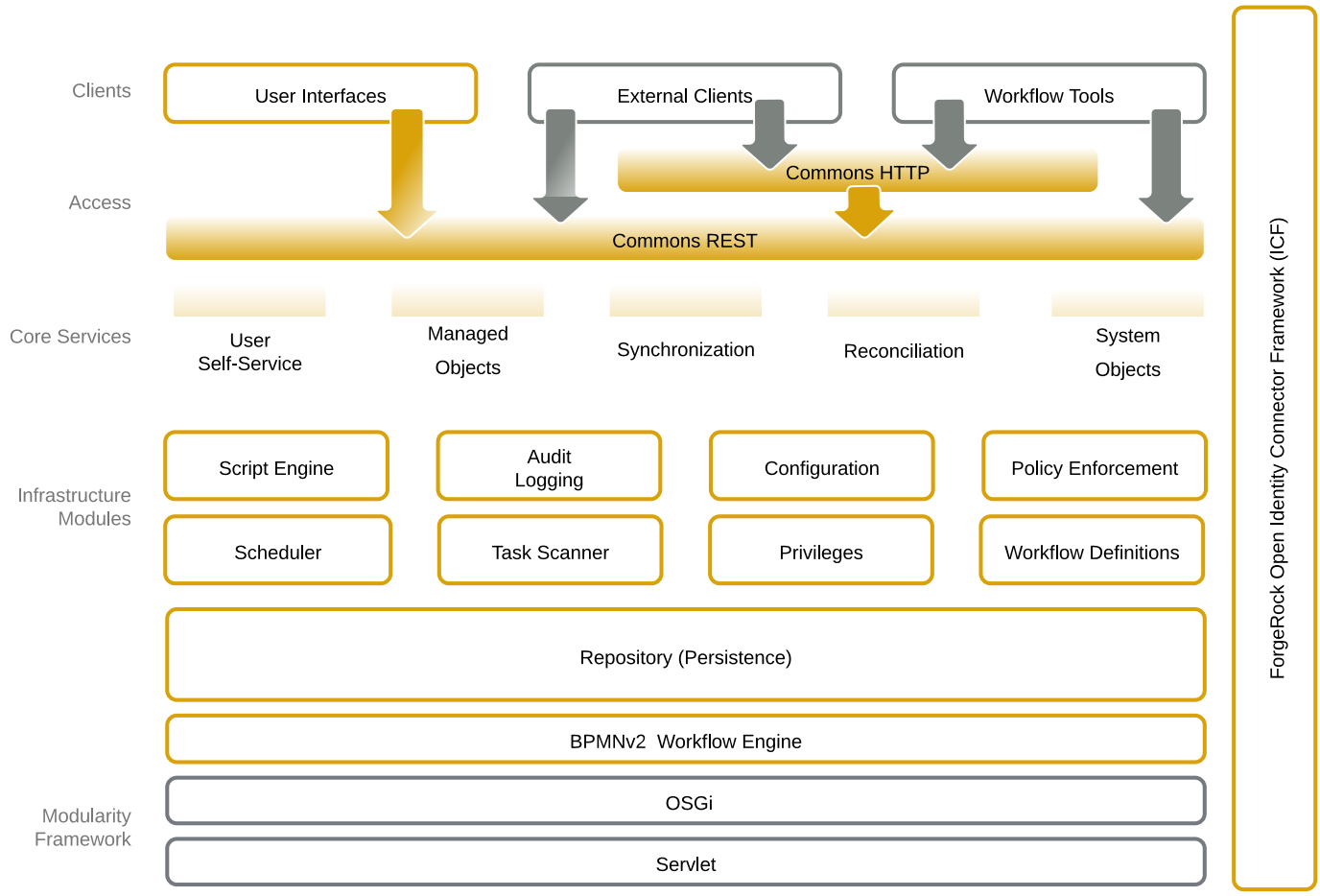


Figure 1. Modular Architecture Overview

The IDM 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 more information, refer to What is OSGi? IDM runs in Apache Felix, an implementation of the OSGi Framework and Service Platform.

Servlet

The Servlet layer provides RESTful HTTP access to the managed objects and services. IDM embeds the Jetty Servlet Container, which can be configured for either HTTP or HTTPS access.

Infrastructure modules

The infrastructure modules provide the underlying features needed for core services:

BPMN 2.0 Workflow Engine

The embedded workflow and business process engine is based on Flowable and the Business Process Model and Notation (BPMN) 2.0 standard.

For more information, refer to Workflow.

Task Scanner

The task scanner performs a batch scan for a specified property, on a scheduled interval, then executes a task when the value of that property matches a specified value.

Scheduler

The scheduler supports Quartz cron triggers and simple triggers. Use the scheduler to trigger regular reconciliations, liveSync, and scripts, to collect and run reports, to trigger workflows, and to perform custom logging.

Script Engine

The script engine is a pluggable module that provides the triggers and plugin points for IDM.

Policy Service

An extensible policy service applies validation requirements to objects and properties, when they are created or updated.

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.

For more information, refer to Configure audit logging.

Repository

The repository provides a common abstraction for a pluggable persistence layer. IDM supports reconciliation and synchronization with several major external data stores in production, including relational databases, LDAP servers, and even flat CSV and XML files.

The repository API uses a JSON-based object model with RESTful principles consistent with the other IDM services. To facilitate testing, IDM includes an embedded instance of ForgeRock Directory Services (DS). In production, you must use a supported repository, as described in Select a repository.

Core services

The core services are the heart of the resource-oriented unified object model and architecture:

Object Model

Artifacts handled by IDM 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.

IDM can serialize and deserialize these structures to and from JSON as required. IDM also exposes a set of triggers and functions that you can define in scripts, which can natively read and modify these JSON-based object model structures.

Managed Objects

A *managed object* is an object that represents the identity-related data managed by IDM. Managed objects are configurable, JSON-based data structures that IDM stores in its pluggable repository. The default managed object configuration includes users and roles, but you can define any kind of managed object, for example, groups or devices.

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

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

System Objects

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

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

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

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

Mappings

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

For more information, refer to Resource mapping.

Reconciliation and Automatic Synchronization

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

Automatic synchronization enables creating, updating, and deleting resources from a source to a target system, either on demand or according to a schedule.

For more information, refer to Synchronization types.

Access layer

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

RESTful Interfaces

IDM provides REST APIs for CRUD operations, for invoking synchronization and reconciliation, and to access several other services.

For more information, refer to the REST API reference.

User Interfaces

User interfaces provide access to most of the functionality available over the REST API.

Server configuration

This chapter describes how IDM loads and stores its configuration, and how to modify it.

The configuration is a combination of `.properties` files, container configuration files, and dynamic configuration objects. Most of the configuration files are stored in your project's `conf/` directory.

Configuration objects

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

> **Note**
>
> If you create custom configuration files, ForgeRock recommends not using spaces or special characters in the filenames, in accordance with the OSGi specification.

The following list describes the single instance configuration objects:

`audit`	Specifies how to log audit events.
`authentication`	Controls REST access.
`cluster`	Defines a clustered IDM instance.
`endpoint`	Controls custom REST endpoints.
`managed`	Defines managed objects and their schemas.
`policy`	Defines the policy validation service.
`process-access`	Defines access to configured workflows.
`repo.repo-type`	Defines the IDM repository; for example, `repo.ds` or `repo.jdbc`.
`router`	Specifies filters to apply for specific operations.

`script`	Defines the parameters that are used when compiling, debugging, and running JavaScript and Groovy scripts.
`sync`	Defines the mappings that IDM uses when it synchronizes and reconciles managed objects.
`ui`	Defines the configurable aspects of the default user interfaces.
`workflow`	Defines the configuration of the workflow engine.

Multiple instance configuration objects

Multiple instance configuration objects correspond to services that can have many instances per installation. Multiple instance configuration objects are named `objectname/instancename`; for example, `provisioner.openicf/csvfile`. *JSON file* views of these configuration objects are named `objectname-instancename.json`, for example, `provisioner.openicf-csvfile.json.`

IDM provides the following multiple instance configuration objects:

- Multiple `schedule` configurations can run reconciliations and other tasks on different schedules.
- Multiple `provisioner.openicf` configurations correspond to connected resources.
- Multiple `servletfilter` configurations can be used for different servlet filters such as the Cross Origin and GZip filters.

Configuration changes

When you change configuration objects, take the following points into account:

- IDM's authoritative configuration source is its repository. Although the JSON files provide a view of the configuration objects, they do not represent the authoritative source.

  Unless you have disabled file writes, IDM updates JSON files after you make configuration changes over REST. You can also edit those JSON files directly. For information on disabling file writes, refer to Disable automatic configuration updates.

- While running, IDM recognizes changes to JSON files. The server *must* be running when you delete configuration objects, even if you do so by editing the JSON files.
- The `openidm.config.file.encoding` property sets the encoding to be used when reading from, or writing to configuration files. The default encoding is UTF-8. Acceptable values include:
  - US-ASCII
  - ISO-8859-1
  - UTF-8
  - UTF-16BE
  - UTF-16LE
  - UTF-16

> **Note**
>
> All configuration files are encoded using UTF-8 by default. If you change the encoding to a different character set, you must re-encode the files before you restart IDM with the new encoding. Failure to do so will result in errors on IDM startup.

- Avoid editing configuration objects directly in the repository. Rather, edit the configuration over the REST API, or in the configuration JSON files to ensure consistent behavior and that operations are logged.
- By default, IDM stores its configuration in the repository. If you remove an IDM instance and do not specifically drop the repository, the configuration remains in effect for a new instance that uses that repository. You can disable this *persistent configuration* in your project's `conf/system.properties` file by setting the following property:

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

  Disabling persistent configuration means that IDM stores its configuration in memory only.

Default REST context

By default, IDM objects are accessible over REST at the context path `/openidm/*`, where `*` indicates the remainder of the context path; for example, `/openidm/managed/user`. You can change the default REST context (`/openidm`) by setting the `openidm.servlet.alias` property in your project's `resolver/boot.properties` file.

The following change to the `boot.properties` file sets the REST context to `/example`:

```
openidm.servlet.alias=/example
```

After this change, objects are accessible at the `/example` context path, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/example/managed/user?_queryFilter=true&_fields=_id"
{
  "result": [
    {
      "_id": "bjensen",
      "_rev": "0000000042b1dcd2"
    },
    {
      "_id": "scarter",
      "_rev": "000000009b54de8a"
    }
  ],
  ...
}
```

To use the admin UI, you must also change the following files, so that the UI is accessible at the new context path:

- In the `/path/to/openidm/ui/admin/default/index.html` file, change the `context`. For example, if your new REST context is `example`, adjust that file as follows:

```
...
<script>
    const context = "/example";
...
</script>
...
```

- In the `/path/to/openidm/ui/admin/default/org/forgerock/openidm/ui/common/util/Constants.js` file, change the value of the `commonConstants.context` property. For example:

```
commonConstants.context = window.context || "/example";
```

Note that changing the REST context impacts the API Explorer. To use the API Explorer with the new REST context, change the `baseUrl` property in the `/path/to/openidm/ui/api/default/index.html` file.

Based on the change to the REST context earlier in this section, you'd set the following:

```
//base URL for accessing the OpenAPI JSON endpoint
var baseURL = '/example/';
```

HTTP I/O buffer

When HTTP I/O requests exceed the memory limit, caching switches to a temporary file. The following lines from `boot.properties` display the default values related to buffer size:

```
# initial size of the in-memory I/O buffer for HTTP requests
#openidm.temporarystorage.initialLength.bytes=8192

# maximum size of the in-memory I/O buffer for HTTP requests
#openidm.temporarystorage.memoryLimit.bytes=65536

# maximum size of the filesystem I/O buffer for HTTP requests, for when memoryLimit is exceeded
#openidm.temporarystorage.fileLimit.bytes=1073741824

# absolute directory path of filesystem I/O buffer for HTTP requests, and uses system property java.io.tmpdir by
default
#openidm.temporarystorage.directory=/var/tmp
```

`openidm.temporarystorage.initialLength.bytes`

Initial size of the memory buffer in bytes.

Default: 8192 bytes (8 KB). Maximum: The value of `openidm.temporarystorage.memoryLimit.bytes`.

`openidm.temporarystorage.memoryLimit.bytes`

Maximum size of the in-memory I/O buffer for HTTP requests. When the memory buffer is full, the content is transferred to a temporary file.

Default: 65536 bytes (64 KB). Maximum: 2147483647 bytes (2 GB).

`openidm.temporarystorage.fileLimit.bytes`

Maximum size of the temporary storage file. If the downloaded file is larger than this value, IDM throws the exception `HTTP 413 Payload Too Large`.

Default: 1073741824 bytes (1 GB). Maximum: 2147483647 bytes (2 GB).

`openidm.temporarystorage.directory`

The absolute directory path of the filesystem I/O buffer for HTTP requests.

Default: The value of the system property `java.io.tmpdir`.

Configure the server over REST

IDM exposes configuration objects under the `/openidm/config` context path.

The optional `waitForCompletion` parameter is available to the `config` endpoint for create, update, and patch requests. Requests to the endpoint with `waitForCompletion=true` delay the response until an OSGi service event confirms the change has been consumed by the corresponding service or the request times out.

The following server properties support additional configuration of the `waitForCompletion` behavior. For more information, refer to Property value substitution.

`openidm.config.waitByDefault`

Default Value: `false`

Specifies whether to wait for the OSGi service event if the `waitForCompletion` parameter is missing from the request.

`openidm.config.waitTimeout`

Default Value: `5000`

The amount of time, in milliseconds, to wait for OSGi service events before timing out.

To list the configuration on the local host, perform a GET request on `http://localhost:8080/openidm/config`.

The following REST call includes excerpts of the default configuration for an IDM instance started with the `sync-with-csv` sample:

```
curl \
--request GET \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
http://localhost:8080/openidm/config
{
  "_id": "",
  "configurations": [
    {
      "_id": "router",
      "pid": "router",
      "factoryPid": null
    },
    {
      "_id": "info/login",
      "pid": "info.f01fc3ed-5871-408d-a5f0-bef00ccc4c8f",
      "factoryPid": "info"
    },
    {
      "_id": "provisioner.openicf/csvfile",
      "pid": "provisioner.openicf.9009f4a1-ea47-4227-94e6-69c345864ba7",
      "factoryPid": "provisioner.openicf"
    },
    {
      "_id": "endpoint/usernotifications",
      "pid": "endpoint.e2751afc-d169-4a23-a88e-7211d340bccb",
      "factoryPid": "endpoint"
    },
    ...
  ]
}
```

Single instance configuration objects are located under `openidm/config/object-name`.

The following example shows the `audit` configuration of the sync-with -csv sample.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
"http://localhost:8080/openidm/config/audit"
{
  "_id": "audit",
  "auditServiceConfig": {
    "handlerForQueries": "json",
    "availableAuditEventHandlers": [
      "org.forgerock.audit.handlers.csv.CsvAuditEventHandler",
      "org.forgerock.audit.handlers.elasticsearch.ElasticsearchAuditEventHandler",
      "org.forgerock.audit.handlers.jms.JmsAuditEventHandler",
      "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
      "org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHandler",
      "org.forgerock.openidm.audit.impl.RepositoryAuditEventHandler",
      "org.forgerock.openidm.audit.impl.RouterAuditEventHandler",
      "org.forgerock.audit.handlers.splunk.SplunkAuditEventHandler",
      "org.forgerock.audit.handlers.syslog.SyslogAuditEventHandler"
    ],
    "filterPolicies": {
      "field": {
        "excludeIf": [],
        "includeIf": []
      }
    },
    "caseInsensitiveFields": [
      "/access/http/request/headers",
      "/access/http/response/headers"
    ]
  },
  "eventHandlers": [
    {
      "class": "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
      "config": {
        "name": "json",
        "enabled": {
          "$bool": "&{openidm.audit.handler.json.enabled|true}"
        },
        "logDirectory": "&{idm.data.dir}/audit",
        "buffering": {
          "maxSize": 100000,
          "writeInterval": "100 millis"
        },
        "topics": [
          "access",
          "activity",
          "sync",
          "authentication",
          "config"
        ]
      }
    },
    {
```

```
      "class": "org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHandler",
      "config": {
        "name": "stdout",
        "enabled": {
          "$bool": "&{openidm.audit.handler.stdout.enabled|false}"
        },
        "topics": [
          "access",
          "activity",
          "sync",
          "authentication",
          "config"
        ]
      }
    },
    {
      "class": "org.forgerock.openidm.audit.impl.RepositoryAuditEventHandler",
      "config": {
        "name": "repo",
        "enabled": {
          "$bool": "&{openidm.audit.handler.repo.enabled|false}"
        },
        "topics": [
          "access",
          "activity",
          "sync",
          "authentication",
          "config"
        ]
      }
    }
  ],
  "eventTopics": {
    "config": {
      "filter": {
        "actions": [
          "create",
          "update",
          "delete",
          "patch",
          "action"
        ]
      }
    },
    "activity": {
      "filter": {
        "actions": [
          "create",
          "update",
          "delete",
          "patch",
          "action"
        ]
      },
      "watchedFields": [],
```

```
      "passwordFields": [
        "password"
      ]
    }
  },
  "exceptionFormatter": {
    "type": "text/javascript",
    "file": "bin/defaults/script/audit/stacktraceFormatter.js"
  }
}
```

Multiple instance configuration objects are found under `openidm/config/object-name/instance-name`.

The following example shows the configuration for the CSV connector from the sync-with-csv sample.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
"http://localhost:8080/openidm/config/provisioner.openicf/csvfile"
{
  "_id": "provisioner.openicf/csvfile",
  "connectorRef": {
    "bundleName": "org.forgerock.openicf.connectors.csvfile-connector",
    "bundleVersion": "[1.5.19.0,1.6.0.0)",
    "connectorName": "org.forgerock.openicf.csvfile.CSVFileConnector"
  },
  "operationTimeout": {
    "CREATE": -1,
    "VALIDATE": -1,
    "TEST": -1,
    "SCRIPT_ON_CONNECTOR": -1,
    "SCHEMA": -1,
    "DELETE": -1,
    "UPDATE": -1,
    "SYNC": -1,
    "AUTHENTICATE": -1,
    "GET": -1,
    "SCRIPT_ON_RESOURCE": -1,
    "SEARCH": -1
  },
  "configurationProperties": {
    "csvFile": "&{idm.instance.dir}/data/csvConnectorData.csv"
  },
  "resultsHandlerConfig": {
    "enableAttributesToGetSearchResultsHandler": true
  },
  "syncFailureHandler": {
    "maxRetries": 5,
    "postRetryAction": "logged-ignore"
  },
  "objectTypes": {
    "account": {
      "$schema": "http://json-schema.org/draft-03/schema",
      "id": "ACCOUNT",
      "type": "object",
      "nativeType": "ACCOUNT",
      "properties": {
        "description": {
          "type": "string",
          "nativeName": "description",
          "nativeType": "string"
        },
        "firstname": {
          "type": "string",
          "nativeName": "firstname",
          "nativeType": "string"
        },
        "email": {
```

```
            "type": "string",
            "nativeName": "email",
            "nativeType": "string"
          },
          "name": {
            "type": "string",
            "required": true,
            "nativeName": "NAME",
            "nativeType": "string"
          },
          "lastname": {
            "type": "string",
            "required": true,
            "nativeName": "lastname",
            "nativeType": "string"
          },
          "mobileTelephoneNumber": {
            "type": "string",
            "required": true,
            "nativeName": "mobileTelephoneNumber",
            "nativeType": "string"
          },
          "roles": {
            "type": "string",
            "required": false,
            "nativeName": "roles",
            "nativeType": "string"
          }
        }
      }
    },
    "operationOptions": {}
}
```

You can change the configuration over REST by using an HTTP PUT or HTTP PATCH request to modify the required configuration object.

The following example uses a PUT request to modify the configuration of the scheduler service, increasing the maximum number of threads that are available for the concurrent execution of scheduled tasks:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--request PUT \
--data '{
  "threadPool": {
    "threadCount": 20
  },
  "scheduler": {
    "executePersistentSchedules": {
      "$bool": "&{openidm.scheduler.execute.persistent.schedules}"
    }
  }
}' \
"http://localhost:8080/openidm/config/scheduler"
{
  "_id": "scheduler",
  "threadPool": {
    "threadCount": 20
  },
  "scheduler": {
    "executePersistentSchedules": {
      "$bool": "&{openidm.scheduler.execute.persistent.schedules}"
    }
  }
}
```

The following example uses a PATCH request to reset the number of threads to their original value.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
  {
    "operation" : "replace",
    "field" : "/threadPool/threadCount",
    "value" : 10
  }
]' \
"http://localhost:8080/openidm/config/scheduler"
{
  "_id": "scheduler",
  "threadPool": {
    "threadCount": 10
  },
  "scheduler": {
    "executePersistentSchedules": {
      "$bool": "&{openidm.scheduler.execute.persistent.schedules}"
    }
  }
}
```

Note

Multi-version concurrency control (MVCC) is not supported for configuration objects so you do not need to specify a revision during updates to the configuration, and no revision is returned in the output.

For more information about using the REST API to update objects, refer to the REST API Reference.

Property value substitution

Property value substitution lets you achieve the following:

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

Property value substitution uses *configuration expressions* to introduce variables into the server configuration. You set configuration expressions as the values of configuration properties. The effective property values can be evaluated in a number of ways. For more information about property evaluation, refer to Expression Resolvers.

Configuration expressions have the following characteristics:

- To distinguish them from static values, configuration expressions are preceded by an ampersand and enclosed in braces. For example: `&{openidm.port.http}`. The configuration token in the example is `openidm.port.http`. The `.` serves as the separator character.
- You can use a default value in a configuration expression by including it after a vertical bar following the token.

  For example, the following expression sets the default HTTP port value to 8080: `&{openidm.port.http|8080}`.

  With this configuration, the server attempts to substitute `openidm.port.http` with a defined configuration token. If no token definition is found, the server uses the default value, `8080`.
- A configuration property can include a mix of static values and expressions.

  For example, suppose `hostname` is set to `ds`. Then, `&{hostname}.example.com` evaluates to `ds.example.com`.
- Configuration token evaluation is recursive.

  For example, suppose `port` is set to `&{port.prefix}389`, and `port.prefix` is set to `2`. Then `&{port}` evaluates to `2389`.

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

> **Important**
>
> Property substitution is *not* available for any configuration not processed by the IDM backend, such as `ui-themeconfig` or any user-supplied configuration.

Expression resolvers

At server startup, expression resolvers evaluate property values to determine the effective configuration. You must define expression values before you start the IDM server that uses them.

When configuration tokens are resolved, the result is always a string. However, you can *coerce* the output type of the evaluated token to match the type that is required by the property. Ultimately, the expression must return the appropriate data type for the configuration property. For example, the `port` property takes an integer. If you set it using an expression, the result of the evaluated expression must be an integer. If the type is wrong, the server fails to start due to a syntax error. For more information about data type coercion, refer to Transforming Data Types.

Expression resolvers can obtain values from the following sources:

Environment variables

You set an environment variable to hold the property value.

For example: `export OPENIDM_PORT_HTTP=8080`.

The environment variable name must be composed of uppercase characters and underscores. The name maps to the expression token as follows:

- Uppercase characters are converted to lowercase.
- Underscores (`_`) are replaced with `.` characters.

In other words, the value of `OPENIDM_PORT_HTTP` replaces `&{openidm.port.http}` in the server configuration.

Java system properties

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

You set a Java system property to hold the value.

Java system property names must match expression tokens exactly. In other words, the value of the `openidm.repo.port` system property replaces `&{openidm.repo.port}` in the server configuration.

Java system properties can be set in a number of ways. One way of setting system properties for IDM servers is to pass them through the `OPENIDM_OPTS` environment variable.

For example: `export OPENIDM_OPTS="-Dopenidm.repo.port=3306"`

System properties can also be declared in your project's `conf/system.properties`.

This example uses property value substitution with a standard system property. The example modifies the audit configuration, changing the `audit.json` file to redirect JSON audit logs to the user's home directory. The `user.home` property is a default Java System property:

```
"eventHandlers" : [
    {
        "class" : "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
        "config" : {
            "name" : "json",
            "logDirectory" : "&{user.home}/audit",
            ...
        }
    },
    ...
]
```

Expression files

You set a key in a `.json` or `.properties` file to hold the value. To use an expression file, set the `IDM_ENVCONFIG_DIRS` environment variable, or the `idm.envconfig.dirs` Java system property as described below. By default, IDM sets `idm.envconfig.dirs` to `&{idm.install.dir}/resolver/`.

The default property resolver file in IDM is `resolver/boot.properties` but you can specify additional files that might hold property values.

Keys in `.properties` files must match expression tokens exactly. In other words, the value of the `openidm.repo.port` key replaces `&{openidm.repo.port}` in the server configuration.

The following example expression properties file sets the repository port:

```
openidm.repo.port=1389
```

JSON expression files can contain nested objects.

JSON field names map to expression tokens as follows:

- The JSON path name matches the expression token.
- The `.` character serves as the JSON path separator character.

The following example JSON expression file uses property value substitution to set the host in the LDAP connector configuration:

```
{
    "openidm" : {
        "provisioner" : {
            "ldap" : {
                "host" : "ds.example.com"
            }
        }
    }
   }
 }
```

To substitute this value in the configuration, the LDAP provisioner file would include the following:

```
{
    ...
    "configurationProperties" : {
        "host" : &{openidm.provisioner.ldap.host|localhost},
        ...
    }
}
```

If the server does not find a configuration token for the host name, it substitutes the default (`localhost`).

To use expression files, set the environment variable, `IDM_ENVCONFIG_DIRS` , or the Java system property, `idm.envconfig.dirs` , to a comma-separated list of the directories containing the expression files.

When reading these files, the server browses the directories in the order specified. It reads all the files with `.json` and `.properties` extensions, and attempts to use them to evaluate expression tokens.

For example, if you define `idm.envconfig.dirs=/directory1,/directory2` and a configuration token is defined in both `directory1` and `directory2` , the resolved value will be the value defined in `directory1` . If the configuration token is defined only in `directory2` , the resolved value will be the value defined in `directory2` .

Note

Using expression files are subject to the following constraints:

- Although IDM scans the directories in a specified order, within a directory IDM scans the files in a nondeterministic order.
- IDM does not scan subdirectories.
- Do *not* define the same configuration token more than once in a file.
  If you define the same property twice in the same file, one definition will be used and the other will be ignored. The server will not throw an error, but because files are scanned in a nondeterministic order, you have no way of knowing which value will be used.
- You cannot define the same configuration token in more than one file in a single directory. The server generates an error in this case.

  Important

  This constraint implies that you cannot have backup `.properties` and `.json` files, in a single directory if they define the same tokens.

- If the same token occurs once in several files that are located in different directories, IDM uses the first value that is read.

Framework configuration properties

You can use the `conf/config.properties` file to override values used by the OSGI framework.

Configuration files

All the properties declared in the `.json` files in your project's `conf/` directory.

Evaluation order of precedence

The following list displays the order of precedence, from greatest to least:

1. Environment variables override system properties, default token settings, and settings in expression files.
2. System properties override default token settings, and any settings in expression files.
3. Default token settings.
4. If `IDM_ENVCONFIG_DIRS` or `idm.envconfig.dirs` is set, the server uses the settings found in expression files.
5. Framework configuration properties.
6. Hardcoded property values.
7. Properties passed to the startup script with options such as: -P, -w, and -s.

Transforming data types

When configuration tokens are resolved, the result is always a string. However, you can transform or *coerce* the output type of the evaluated token to match the type that is required by the property.

You transform a property's data type by setting `$type` before the property value.

The following coercion types are supported:

- array (`$array`)
- boolean (`$bool`)
- decodeBase64 (`$base64:decode`)

  Transforms a base64-encoded string into a decoded string.
- encodeBase64 (`$base64:encode`)

  Transforms a string into a base64-encoded string.
- integer (`$int`)
- list (`$list`)
- number (`$number`)

  This type can coerce integers, doubles, longs, and floats.
- object (`$object`)

  This type can coerce a JSON object such as an encrypted password.

This example JSON expression file sets the value of the port in the LDAP connector configuration:

```
{
    "openidm" : {
        "provisioner" : {
            "ldap" : {
                "port" : 6389
            }
        }
    }
}
```

When the expression is evaluated, the port is evaluated as a `string` value, (which would cause an error). To coerce the port value to an integer, substitute the value in the LDAP provisioner file as follows:

```
{
    ...
    "configurationProperties" : {
        "port" : {
            "$int" : "&{openidm.provisioner.ldap.port|1389}",
        },
        ...
    }
}
```

With this configuration, the server evaluates the LDAP port property to the integer `6389`. If the server does not find a configuration token for the port, it substitutes the default (`1389`).

This example JSON expression file sets a value for the failover servers in an LDAP connector configuration:

```
"openidm" : {
    "provisioner" : {
        "ldap" : {
            "failover" : ["ldap://host1.domain.com:1389", "ldap://host2.domain.com:1389"]
        }
    }
}
```

When the expression is evaluated, the URLs would be evaluated as a single `string`. To coerce the value to an array, substitute the value in the LDAP provisioner file as follows:

```
{
    ...
    "configurationProperties" : {
        "failover" : {
            "$array":"&{openidm.provisioner.ldap.failover}"
        },
        ...
    }
}
```

Note

If you set the failover URLs in a `.properties` file, instead of in a `.json` file, you *must* escape the JSON object. This example sets the failover servers array in the `boot.properties` file:

```
openidm.provisioner.ldap.failover=[\"ldap://host1.domain.com:1389\",\"ldap://host2.domain.com:1389\"]
```

The `$list` function is similar to `$array`, but lets you specify values in a `.properties` file as a list of strings, separated by a comma (`,`).

For example, you could list the LDAP failover servers in `boot.properties` as follows:

```
openidm.provisioner.ldap.failover=ldap://host1.domain.com:1389,ldap://host2.domain.com:1389
```

To coerce the value to an array, your property definition in the LDAP provisioner file would be:

```
{
    ...
    "configurationProperties" : {
        "failover" : {
            "$list":"&{openidm.provisioner.ldap.failover}",
        },
        ...
    }
}
```

This configuration would be converted to:

```
"openidm" : {
    "provisioner" : {
        "ldap" : {
            "failover" : ["ldap://host1.domain.com:1389", "ldap://host2.domain.com:1389"]
        }
    }
}
```

Configuration property value storage

The values of configuration properties that are set explicitly (in `conf/*.json` files) are stored in the repository. You can manage these configuration objects over REST or by using the JSON files themselves.

Properties that use value substitution are stored in the repository as *variables*. You store the *value* of each variable in `.properties` files. You can use different `.properties` files to change the configuration for multiple nodes in a cluster.

The following table shows how specific configuration properties can be set:

Configuration Property Variables

Variable	Description	Environment Variables	System Variables	boot.properties
`idm.install.dir`	Directory of files from unpacked IDM binary	YES	YES	YES
`idm.data.dir`	Working location directory	YES	YES	YES
`idm.instance.dir`	Project directory with IDM configuration files	YES	YES	YES
`idm.envconfig.dirs`	Directory with environment files, including `boot.properties`	YES	YES	

You can access configuration properties in scripts using `identityServer.getProperty()`. For more information, refer to The identityServer Variable.

Limitations of property value substitution

Property value substitution is limited in the following areas:

In the admin UI

Support for property value substitution in the admin UI is limited to the following categories:

- String substitution, where `&{some.property|DefaultValue}`

- Number and integer substitution, including:
  - `"$number" : "&{openidm.port|1234}"`
  - `"$int" : "&{openidm.port|5678}"`
- Base64 substitution, such as: `"$base64:decode" : "&{some.property|YWRtaW4=}"`
- Cryptographic substitution, where for passwords and client secrets, IDM substitutes `"********"` for `$crypto`

In connector configurations

You cannot use property substitution for connector reference (`connectorRef`) properties. For example, the following configuration is *not* valid:

```
"connectorRef" : {
    "connectorName" : "&{connectorName}",
    "bundleName" : "org.forgerock.openicf.connectors.ldap-connector",
    "bundleVersion" : "&{LDAP.BundleVersion}"
    ...
}
```

The `connectorName` must be the precise string from the connector configuration. To specify multiple connector version numbers, use a range of versions. For example:

```
"connectorRef" : {
    "connectorName" : "org.identityconnectors.ldap.LdapConnector",
    "bundleName" : "org.forgerock.openicf.connectors.ldap-connector",
    "bundleVersion" : "[1.5.0.0,2.0.0.0)",
    ...
}
```

HTTP clients

Several IDM modules, such as the external REST service and identity provider service, need to make HTTP(S) requests to external systems.

HTTP client settings can be configured through any expression resolver (in `resolver/boot.properties` , environment variables, or Java system properties). Configuration for specific clients can be set in that client's JSON configuration file. For example, `conf/external.rest.json` configures the external REST service and properties set there override the expression resolvers. For more information on property resolution, refer to Expression Resolvers and Order of Precedence.

You can set the following properties for HTTP clients:

`openidm.http.client.sslAlgorithm`

The cipher to be used when making SSL/TLS connections, for example, `AES` , `CBC` , or `PKCS5Padding` . Defaults to the system SSL algorithm.

`openidm.http.client.socketTimeout`

The TCP socket timeout, in seconds, when waiting for HTTP responses. The default timeout is 10 seconds.

`openidm.http.client.connectionTimeout`

The TCP connection timeout for new HTTP connections, in seconds. The default timeout is 10 seconds.

`openidm.http.client.reuseConnections` *(true or false)*

Specifies whether HTTP connections should be kept alive and reused for additional requests. By default, connections will be reused if possible.

`openidm.http.client.retryRequests` *(true or false)*

Specifies whether requests should be retried if a failure is detected. By default requests will be retried.

`openidm.http.client.maxConnections` *(integer)*

The maximum number of connections that should be pooled by the HTTP client. At most 64 connections will be pooled by default.

`openidm.http.client.hostnameVerifier` *(string)*

Specifies whether the client should check that the hostname to which it has connected is allowed by the certificate that is presented by the server.

The property can take the following values:

- `STRICT` - hostnames are validated
- `ALLOW_ALL` - the external REST service does not attempt to match the URL hostname to the SSL certificate Common Name, as part of its validation process

If you do not set this property, the behavior is to validate hostnames (the equivalent of setting `"hostnameVerifier": "STRICT"`). In production environments, you *should* set this property to `STRICT`.

`openidm.http.client.proxy.uri`

Specifies that the client should make its HTTP(S) requests through the specified proxy server.

`openidm.http.client.proxy.userName`

The username of the account for the specified proxy.

`openidm.http.client.proxy.password`

The password of the account for the specified proxy.

`openidm.http.client.proxy.useSystem` *(true or false)*

If `true`, specifies a system-wide proxy with the JVM system properties, `http.proxyHost`, `http.proxyPort`, and (optionally) `http.nonProxyHosts`.

If `openidm.http.client.proxy.uri` is set, and not empty, that setting overrides the system proxy setting.

Command-line interface

This chapter describes the basic command-line interface (CLI). The CLI includes a number of utilities for managing an IDM instance.

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

```
/path/to/openidm/cli.sh
Using boot properties at /path/to/openidm/resolver/boot.properties
openidm# encrypt ....
```

The command-line utilities run with the security properties defined in your project's `conf/secrets.json` file.

If you run the `cli.sh` command by itself, it opens an IDM-specific shell prompt:

```
openidm#
```

> **Note**
>
> The startup and shutdown scripts are not discussed in this chapter. For information about these scripts, refer to Startup configuration.

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

For a list of subcommands available from the `openidm#` prompt, run the `cli.sh help` command. The `help` and `exit` options shown below are self-explanatory. The other subcommands are explained in the subsections that follow.

```
local:secureHash   Hash the input string.
local:keytool      Export or import a SecretKeyEntry. The Java Keytool does not allow for exporting or
importing SecretKeyEntries.
local:encrypt      Encrypt the input string.
local:validate     Validates all json configuration files in the configuration (default: /conf) folder.
basic:help   Displays available commands.
basic:exit   Exit from the console.
remote:configureconnector   Generate connector configuration.
remote:configexport         Exports all configurations.
remote:configimport         Imports the configuration set from local file/directory.
```

The following options are common to the `configexport`, `configimport`, and `configureconnector` subcommands:

-u or --user USER[:PASSWORD]

Allows you to specify the server user and password. Specifying a username is mandatory. If you do not specify a username, the following error is output to the OSGi console: `Remote operation failed: Unauthorized`. If you do not specify a password, you are prompted for one. This option is used by all three subcommands.

--url URL

The URL of the REST service. The default URL is `http://localhost:8080/openidm/`. This can be used to import configuration files from a remote running IDM instance. This option is used by all three subcommands.

-P or --port PORT

The port number associated with the REST service. If specified, this option overrides any port number specified with the `--url` option. The default port is `8080`. This option is used by all three subcommands.

`configexport`

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

OpenIDM must be running when you execute this command.

Usage is as follows:

```
./cli.sh configexport --user username:password export-location
```

For example:

```
./cli.sh configexport --user openidm-admin:openidm-admin /tmp/conf
```

On Windows systems, the export-location must be provided in quotation marks, for example:

```
C:\openidm\cli.bat configexport --user openidm-admin:openidm-admin "C:\temp\openidm"
```

Configuration objects are exported as `.json` files to the specified directory. The command creates the directory if needed. Configuration files that are present in this directory are renamed as backup files, with a timestamp; for example, `audit.json.2014-02-19T12-00-28.bkp`, and are not overwritten. The following configuration objects are exported:

- The internal repository table configuration (`repo.ds.json` or `repo.jdbc.json`) and the datasource connection configuration, for JDBC repositories (`datasource.jdbc-default.json`)
- The script configuration (`script.json`)
- The log configuration (`audit.json`)
- The authentication configuration (`authentication.json`)
- The cluster configuration (`cluster.json`)
- The configuration of a connected SMTP email server (`external.email.json)`
- Custom configuration information (`info-name.json`)

- The managed object configuration (`managed.json`)
- The connector configuration (`provisioner.openicf-*.json`)
- The router service configuration (`router.json`)
- The scheduler service configuration (`scheduler.json`)
- Any configured schedules (`schedule-*.json`)
- Standard security questions (`selfservice.kba.json)`
- The mapping configuration
- If workflows are defined, the configuration of the workflow engine (`workflow.json`) and the workflow access configuration (`process-access.json`)
- Any configuration files related to the user interface (`ui-*.json`)
- The configuration of any custom endpoints (`endpoint-*.json`)
- The configuration of servlet filters (`servletfilter-*.json`)
- The policy configuration (`policy.json`)

`configimport`

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

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

The subcommand takes the following options:

`-r, --replaceall, --replaceAll`

Replaces the entire list of configuration files with the files in the specified import location.

> **Warning**
>
> This option wipes out the existing configuration and replaces it with the configuration in the import-location. Objects in the existing configuration that are not present in the import-location are deleted.

`--retries` *(integer)*

This option specifies the number of times the command should attempt to update the configuration if the server is not ready.

Default value : 10

`--retryDelay` *(integer)*

This option specifies the delay (in milliseconds) between configuration update retries if the server is not ready.

Default value : 500

Usage is as follows:

```
./cli.sh configimport --user username:password [--replaceAll] [--retries integer] [--retryDelay integer] import-
location
```

For example:

```
./cli.sh configimport --user openidm-admin:openidm-admin --retries 5 --retryDelay 250 --replaceAll /tmp/conf
```

On Windows systems, the import-location must be provided in quotation marks, for example:

```
C:\openidm\cli.bat configimport --user openidm-admin:openidm-admin --replaceAll "C:\temp\openidm"
```

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

`configureconnector`

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

Usage is as follows:

```
./cli.sh configureconnector --user username:password --name connector-name
```

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

```
./cli.sh configureconnector --user openidm-admin:openidm-admin --name myCsvConnector
Executing ./cli.sh…
Starting shell in /path/to/openidm
Mar 26, 2020 06:08:52 PM org.forgerock.openidm.core.FilePropertyAccessor loadProps
0. SSH Connector version 1.5.20.14
1. ServiceNow Connector version 1.5.20.14
2. Scripted SQL Connector version 1.5.20.14
3. Scripted REST Connector version 1.5.20.14
4. Scim Connector version 1.5.20.14
5. Salesforce Connector version 1.5.20.14
6. MSGraphAPI Connector version 1.5.20.14
7. MongoDB Connector version 1.5.20.14
8. Marketo Connector version 1.5.20.14
9. LDAP Connector version 1.5.20.14
10. Kerberos Connector version 1.5.20.14
11. Scripted Poolable Groovy Connector version 1.5.20.14
12. Scripted Groovy Connector version 1.5.20.14
13. GoogleApps Connector version 1.5.20.14
14. Database Table Connector version 1.5.20.14
15. CSV File Connector version 1.5.20.14
16. Adobe Marketing Cloud Connector version 1.5.20.14
17. Exit
Select [0..17]:   15
Edit the configuration file and run the command again. The configuration was saved to
  /path/to/openidm/temp/provisioner.openicf-myCsvConnector.json
```

The basic configuration is saved in a file named `/openidm/temp/provisioner.openicf-connector-name.json`. Edit at least the `configurationProperties` parameter in this file to complete the connector configuration. For example, for a CSV connector:

```
"configurationProperties" : {
    "headerPassword" : "password",
    "csvFile" : "&{idm.instance.dir}/data/csvConnectorData.csv",
    "newlineString" : "\n",
    "headerUid" : "uid",
    "quoteCharacter" : "\"",
    "fieldDelimiter" : ",",
    "syncFileRetentionCount" : 3
}
```

For more information about the connector configuration properties, refer to Configure connectors.

When you have modified the file, run the `configureconnector` command again so that IDM can pick up the new connector configuration:

```
./cli.sh configureconnector --user openidm-admin:openidm-admin --name myCsvConnector
Executing ./cli.sh…
Starting shell in /path/to/openidm
Using boot properties at /path/to/openidm/resolver/boot.properties
Configuration was found and read from: /path/to/openidm/temp/provisioner.openicf-myCsvConnector.json
```

You can now copy the new `provisioner.openicf-myCsvConnector.json` file to your project's `conf/` subdirectory.

You can also configure connectors over the REST interface or through the admin UI. For more information, refer to Configure connectors.

`encrypt`

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

Usage is as follows:

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

If you do not enter the string as part of the command, the command prompts for the string to be encrypted. If you enter the string as part of the command, special characters such as quotation marks, must be escaped.

-j or --json

Indicates that the string to be encrypted is a JSON object, and validates the object. If the object is malformed JSON and you use the `-j` option, the command throws an error. It is easier to input JSON objects in interactive mode. If you input the JSON object on the command-line, the object must be surrounded by quotes, and any special characters, including curly braces, must be escaped. The rules for escaping these characters are fairly complex. For more information, refer to the OSGi specification.

For example:

```
./cli.sh encrypt \
--json '\{\"password\":\"myPassw0rd\"\}'
```

The following example encrypts a normal string value:

```
./cli.sh encrypt \
mypassword
Executing ./cli.sh...
Starting shell in /path/to/openidm
-----BEGIN ENCRYPTED VALUE-----
{
  "$crypto" : {
    "type" : "x-simple-encryption",
    "value" : {
      "cipher" : "AES/CBC/PKCS5Padding",
      "stableId" : "openidm-sym-default",
      "salt" : "vdz6bUztiT6QsExNrZQAEA==",
      "data" : "RgMLRbX0guxF80nwrtaZkkoFFGqSQdNWF7Ve0zS+N1I=",
      "keySize" : 16,
      "purpose" : "idm.config.encryption",
      "iv" : "R9w1TcWfbd9FPmOjfvMhZQ==",
      "mac" : "9pXtSKAt9+dO3Mu0NlrJsQ=="
    }
  }
}
------END ENCRYPTED VALUE------
```

The following example prompts for a JSON object to be encrypted:

```
./cli.sh encrypt --json
Using boot properties at /path/to/openidm/resolver/boot.properties
Enter the Json value

> Press ctrl-D to finish input
Start data input: {"password":"myPassw0rd"}
^D
-----BEGIN ENCRYPTED VALUE-----
{
  "$crypto" : {
    "type" : "x-simple-encryption",
    "value" : {
      "cipher" : "AES/CBC/PKCS5Padding",
      "stableId" : "openidm-sym-default",
      "salt" : "vdz6bUztiT6QsExNrZQAEA==",
      "data" : "RgMLRbX0guxF80nwrtaZkkoFFGqSQdNWF7Ve0zS+N1I=",
      "keySize" : 16,
      "purpose" : "idm.config.encryption",
      "iv" : "R9w1TcWfbd9FPmOjfvMhZQ==",
      "mac" : "9pXtSKAt9+dO3Mu0NlrJsQ=="
    }
  }
}
------END ENCRYPTED VALUE------
```

`secureHash`

The `secureHash` subcommand hashes an input string, or JSON object, using the specified hash algorithm configuration. Use this subcommand to hash password values, or other sensitive data, to be stored in the repository. The hashed value is output to standard output and provides details of the algorithm configuration that was used to hash the data.

Usage is as follows:

```
/path/to/openidm/cli.sh secureHash --algorithm --config [--json] string
```

-a or --algorithm

Specifies the hash algorithm to use.

-c or --config

Lets you provide additional hashing configuration options, as a JSON object. For a list of supported hash algorithms and their configuration, refer to Salted Hash Algorithms.

-j or --json

Indicates that the string to be encrypted is a JSON object, and validates the object. If the object is malformed JSON and you use the `-j` option, the command throws an error. It is easier to input JSON objects in interactive mode. If you input the JSON object on the command-line, the object must be surrounded by quotes, and any special characters, including curly braces, must be escaped. The rules for escaping these characters are fairly complex. For more information, refer to the OSGi specification.

For example:

```
/path/to/openidm/cli.sh secureHash \
--algorithm SHA-384 \
--json '\{\"password\":\"myPassw0rd\"\}'
```

If you do not enter the string as part of the command, the command prompts for the string to be hashed. If you enter the string as part of the command, any special characters, for example quotation marks, must be escaped.

The following example hashes a password value (`mypassword`) using the `PBKDF2` algorithm:

```
/path/to/openidm/cli.sh secureHash \
--algorithm PBKDF2 \
--config '{\"hashLength\":16\,\"saltLength\":16\,\"iterations\":20000\,\"hmac\":\"SHA3-256\"}' \
"mypassword"
Executing ./cli.sh…
Starting shell in /path/to/openidm
…
-----BEGIN HASHED VALUE-----
{
  "$crypto" : {
    "value" : {
      "algorithm" : "PBKDF2",
      "data" : "9/1IIaAVxAMFdCzlMGtkXMmotKqBafIdx2KFUeKHX0k=",
      "config" : {
        "hashLength" : 16,
        "saltLength" : 16,
        "iterations" : 20000,
        "hmac" : "SHA3-256"
      }
    },
    "type" : "salted-hash"
  }
}
------END HASHED VALUE------
```

The following example prompts for a JSON object to be hashed:

```
/path/to/openidm/cli.sh secureHash --algorithm SHA-384 --json
Executing ./cli.sh…
Executing ./cli.sh…
Starting shell in /path/to/openidm
Nov 14, 2017 1:24:26 PM org.forgerock.openidm.core.FilePropertyAccessor loadProps
INFO: Using properties at /path/to/openidm/resolver/boot.properties
Enter the Json value

> Press ctrl-D to finish input
Start data input: {"password":"myPassw0rd"}
^D
-----BEGIN HASHED VALUE-----
{
  "$crypto" : {
    "value" : {
      "algorithm" : "SHA-384",
      "data" : "7Caabx7d+vOZ7d3VMwdQObQJdTQ3uGOItsX5AwR4ViygUfARR/XuxRIBQt1LRq58ZOQXFwuw+3rvzK7Kld8pSg=="
    },
    "type" : "salted-hash"
  }
}
------END HASHED VALUE------
```

keytool

The `keytool` subcommand exports or imports secret key values.

The Java `keytool` command enables you to export and import public keys and certificates, but not secret or symmetric keys. The IDM `keytool` subcommand provides this functionality.

Usage is as follows:

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

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

```
./cli.sh keytool --export openidm-sym-default
Executing ./cli.sh…
Starting shell in /home/idm/openidm
Use KeyStore from: /openidm/security/keystore.jceks
Please enter the password:
[OK] Secret key entry with algorithm AES
AES:606d80ae316be58e94439f91ad8ce1c0
```

The default keystore password is `changeit`. For security reasons, you *must* change this password in a production environment. For information about changing the keystore password, refer to Default keystore.

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

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

If a secret key with that name already exists, IDM returns the following error:

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

validate

The `validate` subcommand validates all .json configuration files in your project's `conf/` directory.

Usage is as follows:

```
./cli.sh validate
Executing ./cli.sh
Starting shell in /path/to/openidm
Using boot properties at /path/to/openidm/resolver/boot.properties
.....................................................................
[Validating] Load JSON configuration files from:
[Validating]    /path/to/openidm/conf
[Validating] audit.json ................................ SUCCESS
[Validating] authentication.json ....................... SUCCESS
    ...
[Validating] sync.json ................................. SUCCESS
[Validating] ui-configuration.json ..................... SUCCESS
[Validating] ui-countries.json ......................... SUCCESS
[Validating] workflow.json ............................. SUCCESS
```

IDM user interface

IDM provides two customizable, browser-based user interfaces: an Administrative User Interface (admin UI) and an End User interface (End User UI).

The admin UI provides a graphical interface for most aspects of the IDM configuration. If IDM is installed on the local system, access the admin UI at `https://localhost:8443/admin`. The admin UI lets you configure and manage users and roles, set up synchronization between resources, configure connectors, and more.

The End User UI provides role-based access to tasks based on BPMN2 workflows, and allows users to manage certain aspects of their own accounts, including configurable self-service registration. End users access the UI at a URL you specify. As an administrator, if IDM is installed on the local system, you can access the End User UI at `https://localhost:8443/`. All users, including `openidm-admin`, can change their password through the End User UI. The End User UI is described in Self-service end user UI.

> **Caution**
>
> Browser ad blocker extensions can inadvertently block some UI functionality, particularly if your configuration includes strings such as `ad`. For example, a connection to an Active Directory server might be configured at the endpoint `system/ad`. To avoid problems related to blocked UI functionality, remove the extension, or configure a safelist to ensure access to the targeted endpoints.

Manage dashboards

Dashboards let you make shortcuts to frequently-required tasks. The **Quick Start** dashboard displays by default when you log in to the admin UI. You can create additional dashboards, or add and remove widgets from the existing dashboards.

Default dashboards

To display all configured dashboards, select **Dashboards > Manage Dashboards**. The following dashboards are provided by default.

Quick start dashboard

Quick Start cards support one-click access to common administrative tasks:

Add Connector

Configure connections to external resources.

Create Mapping

Configure mappings to synchronize objects between resources.

Manage Roles

Set up provisioning or authorization roles.

Add Device

Configure managed objects, including users, groups, roles, and devices.

Configure Registration

Configure user self-registration.

Configure Password Reset

Configure user self-service password reset.

Manage Users

Manage users in the repository.

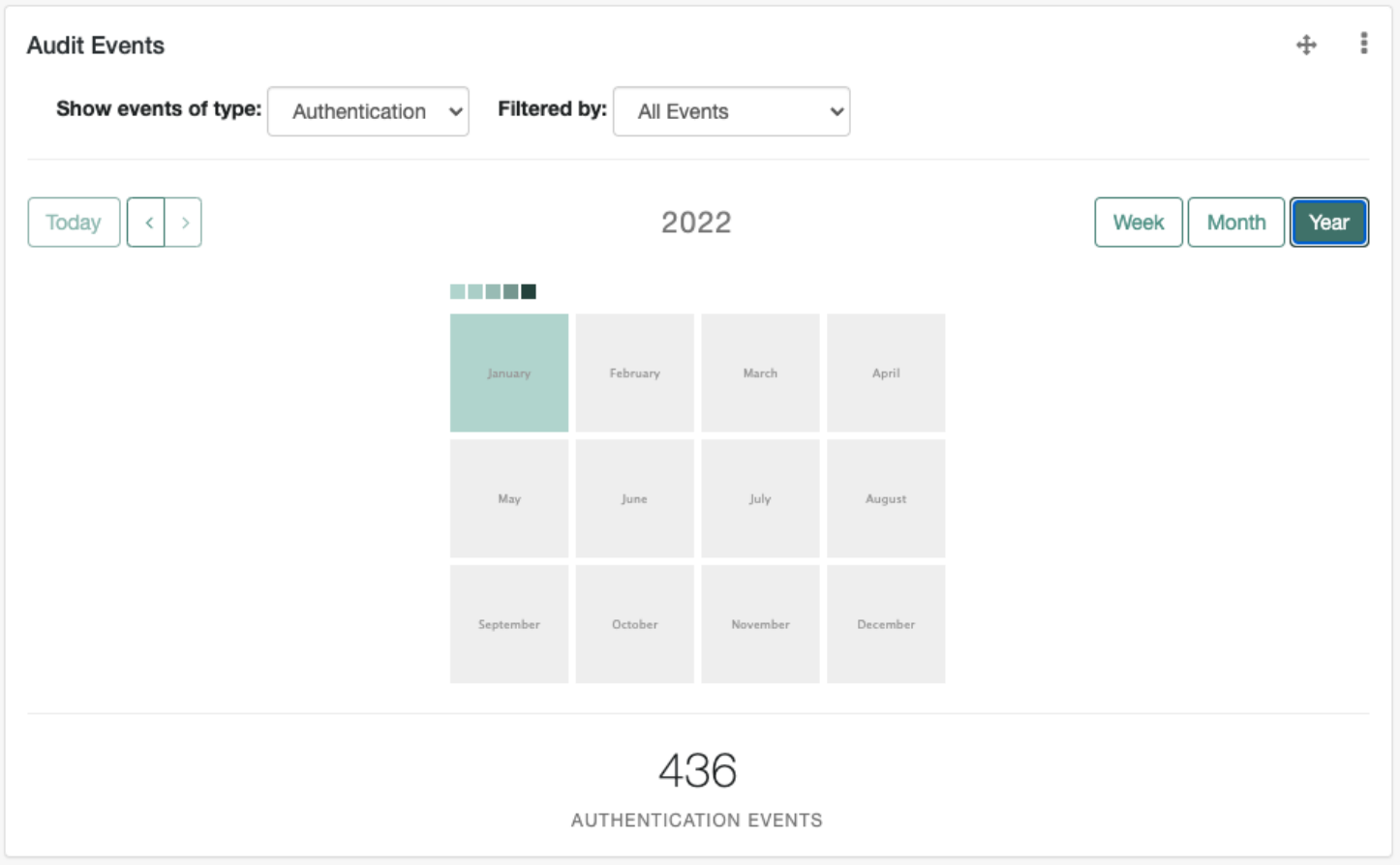
Configure System Preferences

Configure server settings for Audit, Workflow, and more.

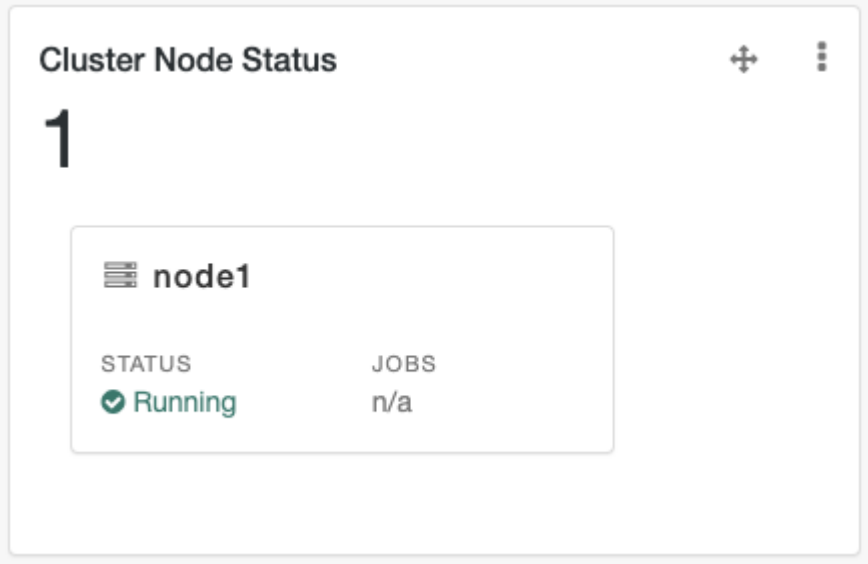
System monitoring dashboard

The System Monitoring Dashboard includes information about:

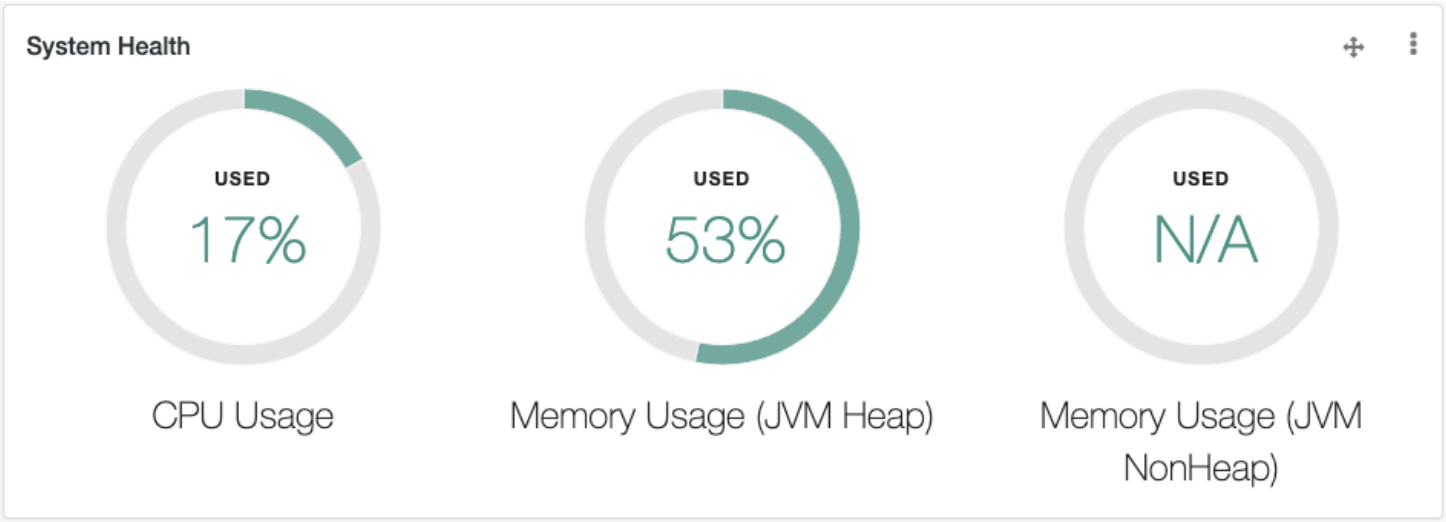
Includes information on audit data, organized by date.



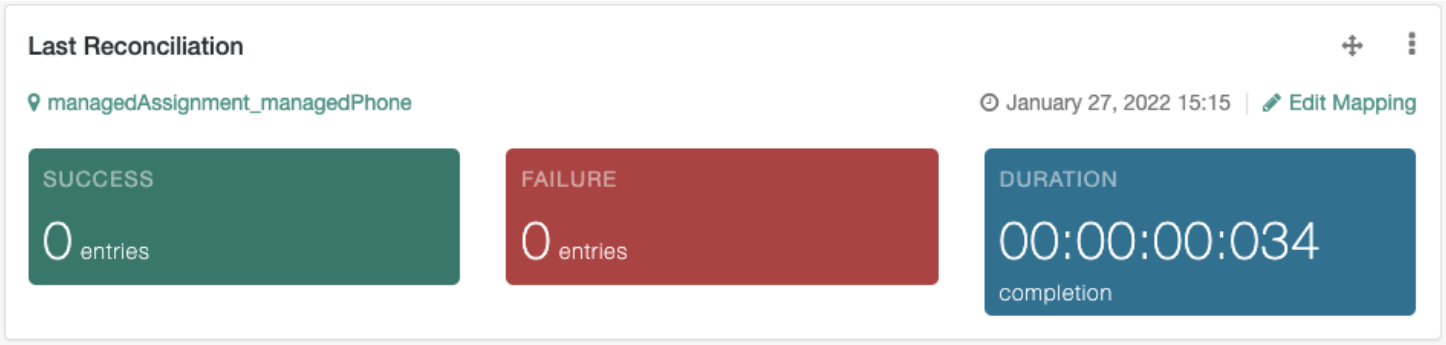
Includes information on cluster nodes.



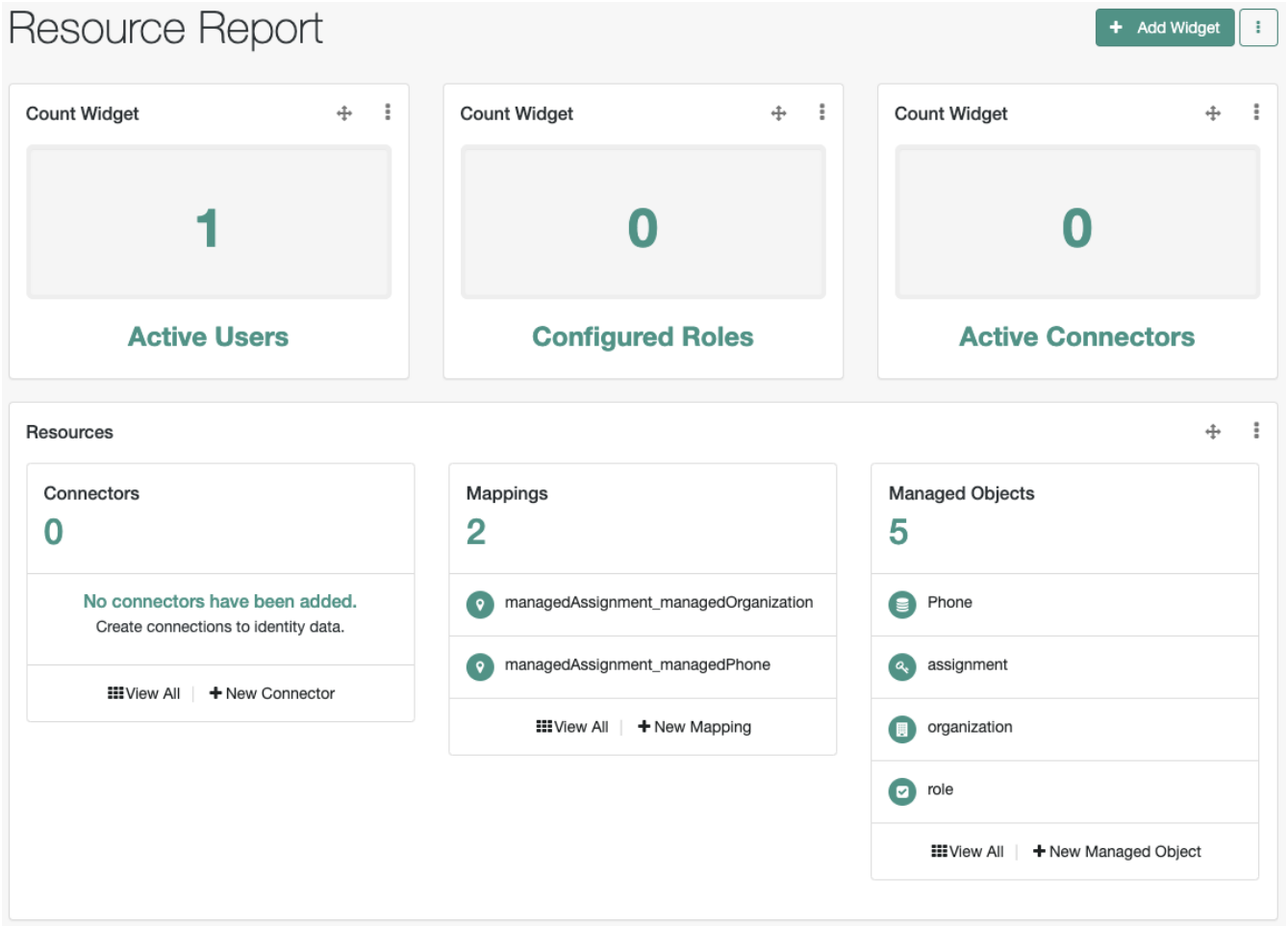
Includes information on system resource usage.



Includes data from the most recent data reconciliation.

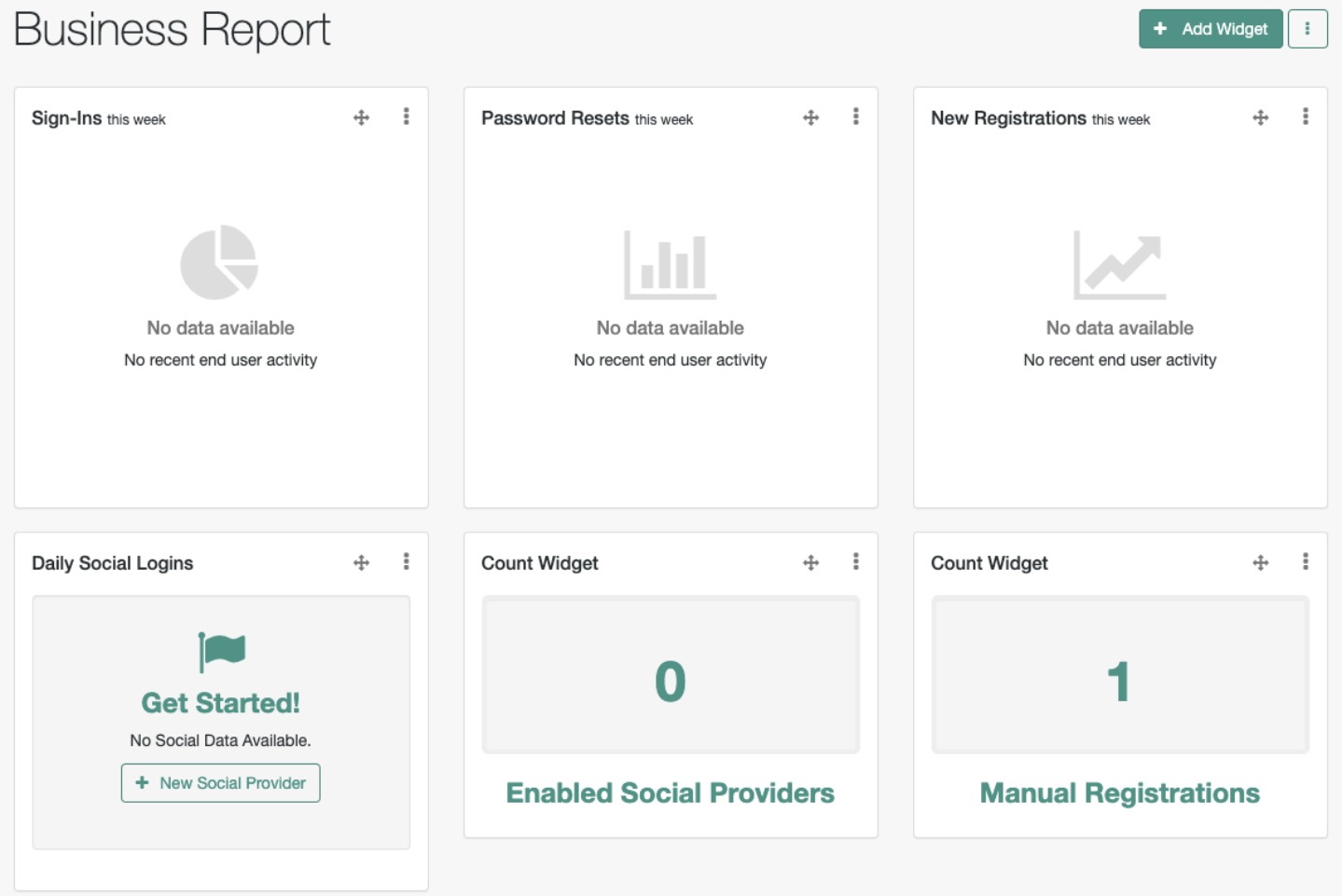


Resource report



- The Resource Report includes widgets that show the number of active users, configured roles, and active connectors.
- The Resources widget shows all configured connectors, mappings, and managed object types.

Business report



The Business Report includes widgets related to login and registration activity.

Custom dashboards

You can set up additional dashboards for customized views of the admin UI.

Create a new dashboard

To create a new dashboard, select **Dashboards > New Dashboard**. Enter a dashboard name and select whether this dashboard should be the default board that is displayed when you load the admin UI.

For a customized view of the admin UI, select **Widgets** as the **Dashboard Type**, click **Create Dashboard**, and add the widgets that you want exposed in that view.

You can also customize the view by starting with an existing dashboard. In the upper-right corner of the UI, next to the **Add Widget** button, click the **overflow menu ⋮ > widget**, and select **Rename** or **Duplicate**.

Add and move widgets

To add a widget to a dashboard, click **Add Widget** and select the widget type. Widgets are grouped in categories. Scroll down to the category of the widget you want to add.

To change the position of a widget on a dashboard, click and drag the move button ✣.

To add a new **Quick Start** widget, select the **overflow menu ⋮ > widget** in the upper right corner of the widget, and click **Settings**.

To embed an admin UI sub-widget in the **Quick Start** widget, specify the destination URL. If you are linking to a specific page in the admin UI, the destination URL can be the part of admin UI address. For example, to create a quick start link to the **Audit Configuration** tab, at `{secureHostname}/admin/#settings/audit/`, enter `#settings/audit` in the destination URL text box.

Any changes to the dashboards are persisted in your project's `conf/ui-dashboard.json` file, which has the following properties:

admin UI Widget Properties in `ui-dashboard.json`

Property	Values	Description
`name`	String	Dashboard name.
`isDefault`	`true` or `false`	Default dashboard. You can only set one default.
`widgets`	Different attributes based on `type`	Attributes that define the widget.
`type`	`lastRecon`, `resourceList`, `quickStart`, `userRelationship`	Widget type.
`size`	`x-small`, `small`, `medium`, or `large`	Width of widget, based on a 12-column grid system, where x-small=4, small=6, medium=8, and large=12. For more information, refer to Bootstrap CSS.
`barchart`	`true` or `false`	Reconciliation bar chart; applies only to the **Last Reconciliation** widget.

Admin UI widgets

The following tables list the available widgets:

admin UI reporting widgets

Name	Description
Audit Events	Graphical display of audit events.
Count Widget	A *count* widget that provides an instant display of the number of specific objects; for example, active managed users, and enabled social providers.
Dropwizard Table With Graph	Does not appear in the list of widgets unless metrics are active.
Graph Widget	Provides a graphical view of a specific managed resource; for example, managed users, based on some metric.

Name	Description
Last Reconciliation	Shows statistics from the most recent reconciliation, shown on the System Monitoring dashboard.
New Registrations	The number of users that have self-registered that week. To display data using this widget, you must enable user self-registration.
Password Resets	The number of password resets that week. To display data using this widget, you must enable password reset.
Resources	Connectors, mappings, managed objects; shown in Administration dashboard.
Sign-Ins	The number of managed users that have signed in to the service that week.

Social widgets

Name	Description
Daily Social Logins	Graphical display of logins through social identity providers.
Social Registration (year)	Graphical display of social registrations over the past year.

System status widgets

Name	Description
Cluster Node Status	Lists the instances in a cluster, with their status.

Utility widgets

Name	Description
Quick Start	Links to common tasks; shown in the Administration dashboard.
Identity Relationships	Graphical display of relationships between identities.
Managed Objects Relationship Diagram	Graphical diagram with connections between managed object properties; also refer to View the Relationship Configuration in the UI.

Customize the admin UI

This section shows you how to customize the admin UI for your deployment.

The default admin UI configuration files are located in `openidm/ui/admin/default`. To customize the UI, copy this directory to `openidm/ui/admin/extension`:

```
cd /path/to/openidm/ui/admin
cp -r default/. extension
```

You can now edit the custom files in the `extension` subdirectory. The admin UI templates in `openidm/ui/admin/default/templates` can help you get started.

Default UI subdirectories

The admin UI config files are located in separate subdirectories:

`config/`

Top-level configuration directory of JavaScript files. Customizable subdirectories include `errorhandlers/` with HTTP error messages and `messages/` with info and error messages. For actual messages, refer to the `translation.json` file in the `locales/en/` subdirectory.

`css/` *and* `libs/`

If you use a different bootstrap theme, replace the files in these directories and subdirectories.

`fonts/`

The font files in this directory are based on the Font Awesome CSS toolkit.

`images/` *and* `img/`

IDM uses the image files in these directories. You can replace these images with your own.

`locales/`

The `translation.json` file (in the `en/` subdirectory by default) contains the UI labels and messages.

`org/`

Source files for the admin UI.

`partials/`

Includes partial components of HTML pages in the admin UI, for assignments, authentication, connectors, dashboards, email, basic forms, login buttons, and so on.

`templates/`

The HTML templates for various UI pages. Note that these files are used by the UI. Do not change the template files in `ui/admin/default/`.

Customize the UI theme

You can configure a few features of the UI in the `ui-themeconfig.json` file in your project's `conf` directory. However, to change most theme-related features of the UI, you must copy target files to the appropriate `extension` subdirectory, and then modify them as discussed in Customize the admin UI.

By default, the UI reads the stylesheets and images from the `openidm/ui/admin/default` directory. Do not modify the files in this directory. Your changes may be overwritten the next time you update or even patch your system.

To customize your UI, first set up matching subdirectories in (`openidm/ui/admin/extension`). For example, `openidm/ui/admin/extension/libs` and `openidm/ui/admin/extension/css` .

You might also need to update the `"stylesheets"` listing in the `ui-themeconfig.json` file for your project, in the `project-dir/conf` directory.

```
"stylesheets" : [
    "css/bootstrap-3.4.1-custom.css",
    "css/structure.css",
    "css/theme.css"
],
```

The default stylesheets contain:

- `bootstrap-3.4.1-custom.css` —Includes custom settings that you can get from various Bootstrap configuration sites, such as the Bootstrap Customize and Download site. This site lets you upload a `config.json` file that makes it easier to create a customized Bootstrap file. The ForgeRock version of this file is in `ui/admin/default/css/common/structure/` . You can use this file as a starting point for your customization.
- `structure.css` —For configuring structural elements of the UI.
- `theme.css` —Includes customizable options for UI themes such as colors, buttons, and navigation bars.

To set up custom versions of these files, copy them to the `extension/css` subdirectories.

Change the default logo

The default UI logo is in `openidm/ui/admin/default/images` . To change the logo, place your custom image in the `openidm/ui/admin/extension/images` directory. You should see the changes after refreshing your browser.

To specify a different file name, or to control the size, and other properties of the logo image file, adjust the `logo` property in the UI theme configuration file for your project `(conf/ui-themeconfig.json)` .

The following change to the UI theme configuration file points to an image file named `example-logo.png` , in the `openidm/ui/admin/extension/images` directory:

```
...
"loginLogo" : {
     "src" : "images/example-logo.png",
     "title" : "Example.com",
     "alt" : "Example.com",
     "height" : "104px",
     "width" : "210px"
},
...
```

Refresh your browser window for the new logo to appear.

Create project-specific themes

You can create different UI themes for projects and then point a particular UI instance to a defined theme on startup. To create a complete custom theme, follow these steps:

1. Shut down the IDM instance.
2. Copy the entire default admin UI theme to an accessible location. For example:

   ```
   cp -r /path/to/openidm/ui/admin/default /path/to/openidm/admin-project-theme
   ```

3. In the copied theme, modify the required elements, as described in the previous sections. Note that nothing is copied to the extension folder in this case—changes are made in the copied theme.

   In the `conf/ui.context-admin.json` file, modify the values for `defaultDir` and `extensionDir` to the directory with your `new-project-theme`:

   ```
   {
       "enabled" : true,
       "cacheEnabled" : true,
       "urlContextRoot" : "/",
       "defaultDir" : "&{idm.install.dir}/ui/admin/default",
       "extensionDir" : "&{idm.install.dir}/ui/admin/extension",
       "responseHeaders" : {
           "X-Frame-Options" : "DENY"
       }
   }
   ```

4. Restart the server.
5. Relaunch the UI in your browser.

   The UI displays with the new custom theme.

Set custom response headers

You can specify custom response headers for your UI by using the `responseHeaders` property in UI context configuration files such as `conf/ui.context-admin.json`. For example, the `X-Frame-Options` header is a security measure used to prevent a web page from being embedded within the frame of another page. For more information about response headers, refer to the MDN page on HTTP Headers.

Because the `responseHeaders` property is specified in the configuration file for each UI context, you can set different custom headers for different UIs. For example, you might set different security headers included for the Admin and End User UIs.

Disable the Admin and End User UIs

The UIs are packaged as separate bundles that can be disabled in the configuration before server startup.

To disable the registration of the UI servlets, edit the `project-dir/conf/ui.context-ui.json` files, setting the `enabled` property to false. For example, to disable the End User UI, set `"enabled" : false,` in `project-dir/conf/ui.context-enduser.json`

Protect custom static web resources

By default, custom static web resources are publicly accessible. To require authentication to access these resources, add the following line to the applicable `project-dir/conf/ui.context-ui.json` files:

```
"authEnabled" : true,
```

Cache static UI files in memory

By default, static UI files are cached in memory for all but the API Explorer. To disable the caching of specific UI files, set `"cacheEnabled": false` in the applicable `project-dir/conf/ui.context-ui.json` file.

When caching is enabled, all static files are cached in-memory, but off-heap, when they are first accessed. When caching is disabled, the files are streamed from disk.

> **Caution**
>
> UI asset directories that contain large files should not enable caching. You can reduce RAM usage by disabling caching where it is not required.

Reset user passwords

When working with end users, administrators frequently have to reset their passwords. You can do so directly, through the admin UI. Alternatively, you can configure an external system for that purpose, or set up password reset, as described in Password reset.

Change user passwords through the admin UI

1. From the navigation bar, click **Manage > User**, and click a user.
2. Click the **Password** tab, and change the password.

Use an external password reset system

By default, the Password Reset mechanism is handled within IDM. You can reroute Password Reset in the event that a user has forgotten their password, by specifying an external URL to which Password Reset requests are sent. Note that this URL applies to the Password Reset link on the login page only, not to the security data change facility that is available after a user has logged in.

To set an external URL to handle Password Reset, set the `passwordResetLink` parameter in `conf/ui-configuration.json`. The following example sets the `passwordResetLink` to `https://accounts.example.com/account/reset-password`:

```
passwordResetLink: "https://accounts.example.com/reset-password"
```

The `passwordResetLink` parameter takes either an empty string as a value (which indicates that no external link is used) or a full URL to the external system that handles Password Reset requests.

> **Note**
>
> External Password Reset and security questions for internal Password Reset are mutually exclusive. Therefore, if you set a value for the `passwordResetLink` parameter, users will not be prompted with any security questions, regardless of the setting of the `securityQuestions` parameter.

Object modeling

Guide to creating and managing objects in ForgeRock® Identity Management.

IDM provides a default schema for typical managed object types, such as users, roles, and groups, but does not control the structure of objects that you store in the repository. This section shows you how to change and add to the managed object schema, how to establish relationships between objects, and how to use policies to validate objects. You will also learn how to access IDM objects using queries.

Quick start

Managed objects

Learn about the IDM architecture, component modules, and services.

Users

Understand the default user object.

Relationships

Configure relationships between object types.

Roles

Learn about the role object, a specific relationship type.

Policies

Apply validation requirements to objects and properties.

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

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

Managed objects

These topics describe how to work with managed object types. For more information about the IDM object model, refer to Data models and objects reference.

- Define the Schema
- Create and modify object types
- Virtual properties
- Run scripts on managed objects
- Track user metadata

Define the schema

Managed objects and their properties are defined in the managed object configuration.

The default managed object configuration is not a comprehensive list of all the properties that can be stored in IDM. If you use a generic object mapping, you can create a managed object with any arbitrary property, and that property will be stored in IDM. However, if you create an object with properties that are not defined in the managed object configuration, those properties are not visible in the UI. In addition, you won't be able to configure the "sub-properties" that are described in the following section.

Important

- The admin UI depends on the presence of specific core schema elements, such as users, roles, and assignments (and the default properties nested within them). If you remove such schema elements, and you use the admin UI to configure IDM, you must modify the admin UI code accordingly. For example, if you remove the entire `assignment` object from the managed object configuration, the UI will throw exceptions wherever it queries this schema element.
- Managed object properties that contain an underscore (`_`) are reserved for internal use. Do not create new properties that contain underscores, and do not include these properties in update requests.

Create and modify object types

If the managed object types provided in the default configuration are not sufficient for your deployment, you can create new managed object types. The easiest way to create a new managed object type is to use the admin UI, as follows:

1. Select **Configure > Managed Objects > New Managed Object**.
2. On the **New Managed Object** page, enter a name and readable title for the object, make optional changes, as necessary, and click **Save**. The readable title specifies what the object will be called in the UI.
3. On the **Properties** tab, specify the schema for the object type (the properties that make up the object).
4. On the **Scripts** tab, specify any scripts that will be applied on events associated with that object type. For example, scripts that will be run when an object of that type is created, updated, or deleted.

You can also create a new managed object type by editing your managed object configuration.

```
{
    "name": "Phone",
    "schema": {
        "$schema": "http://forgerock.org/json-schema#",
        "type": "object",
        "properties": {
            "brand": {
                "description": "The supplier of the mobile phone",
                "title": "Brand",
                "viewable": true,
                "searchable": true,
                "userEditable": false,
                "policies": [],
                "returnByDefault": false,
                "pattern": "",
                "isVirtual": false,
                "type": [
                    "string",
                    "null"
                ]
            },
            "assetNumber": {
                "description": "The asset tag number of the mobile device",
                "title": "Asset Number",
                "viewable": true,
                "searchable": true,
                "userEditable": false,
                "policies": [],
                "returnByDefault": false,
                "pattern": "",
                "isVirtual": false,
                "type": "string"
            },
            "model": {
                "description": "The model number of the mobile device, such as 6 plus, Galaxy S4",
                "title": "Model",
                "viewable": true,
                "searchable": false,
                "userEditable": false,
                "policies": [],
                "returnByDefault": false,
                "pattern": "",
                "isVirtual": false,
                "type": "string"
            }
        },
        "required": [],
        "order": [
            "brand",
            "assetNumber",
            "model"
        ]
    }
}
```

Every managed object type has a `name` and a `schema` that describes the properties associated with that object. The `name` can only include the characters `a-z`, `A-Z`, `0-9`, and `_` (underscore). You can add any arbitrary properties to the schema.

> **Tip**
>
> Avoid using the dash character in property names (like `last-name`) because dashes in names make JavaScript syntax more complex. Rather use "camel case" (`lastName`). If you cannot avoid dash characters, write `source['last-name']` instead of `source.last-name` in your JavaScript.

A property definition typically includes the following fields:

`title`

The name of the property, in human-readable language, used to display the property in the UI.

`description`

A brief description of the property.

`viewable`

Specifies whether this property is viewable in the object's profile in the UI. Boolean, `true` or `false` (`true` by default).

`searchable`

Specifies whether this property can be searched in the UI. A searchable property is visible within the Managed Object data grid in the End User UI.

For a property to be searchable in the UI, it *must be indexed* in the repository configuration. For information on indexing properties in a repository, refer to Object mappings.

Boolean, `true` or `false` (`false` by default).

`userEditable`

Specifies whether users can edit the property value in the UI. This property applies in the context of the End User UI, where users are able to edit certain properties of their own accounts. Boolean, `true` or `false` (`false` by default).

`isProtected`

Specifies whether reauthentication is required if the value of this property changes.

For certain properties, such as passwords, changing the value of the property should force an end user to reauthenticate. These properties are referred to as *protected properties*. Depending on how the user authenticates (which authentication module is used), the list of protected properties is added to the user's security context. For example, if a user logs in with the login and password of their managed user entry (`MANAGED_USER` authentication module), their security context will include this list of protected properties. The list of protected properties is not included in the security context if the user logs in with a module that does not support reauthentication (such as through a social identity provider).

`pattern`

Any specific pattern to which the value of the property must adhere. For example, a property whose value is a date might require a specific date format.

`policies`

Any policy validation that must be applied to the property. For more information on managed object policies, refer to Default policy for managed objects.

`required`

Specifies whether the property must be supplied when an object of this type is created. Boolean, `true` or `false`.

> **Important**
>
> The `required` policy is assessed only during object creation, not when an object is updated. You can effectively bypass the policy by updating the object and supplying an empty value for that property. To prevent this inconsistency, set both `required` and `notEmpty` to `true` for required properties. This configuration indicates that the property must exist, and must have a value.

`type`

The data type for the property value; can be `string`, `array`, `boolean`, `integer`, `number`, `object`, `ResourceCollection`, or `null`.

> **Note**
>
> If any user might not have a value for a specific property (such as a `telephoneNumber`), you must include `null` as one of the property *types*. You can set a null property type in the admin UI (**Configure > Managed Objects > User**, select the property, and under the **Details** tab, **Advanced Options**, set `Nullable` to `true`).
> You can also set a null property type in your managed object configuration by setting `"type" : '[ "string","null" ]'` for that property (where `string` can be any other valid property type. This information is validated by the policy service, as described in Validate Managed Object Data Types.
> If you're configuring a data `type` of `array` through the admin UI, you're limited to two values.

`isVirtual`

Specifies whether the property takes a static value, or whether its value is calculated "on the fly" as the result of a script. Boolean, `true` or `false`.

`returnByDefault`

For non-core attributes (virtual attributes and relationship fields), specifies whether the property will be returned in the results of a query on an object of this type *if it is not explicitly requested*. Virtual attributes and relationship fields are not returned by default. Boolean, `true` or `false`. When the property is in an array within a relationship, always set to `false`.

`relationshipGrantTemporalConstraintsEnforced`

For attributes with relationship fields. Specifies whether this relationship should have temporal constraints enforced. Boolean, `true` or `false`. For more information about temporal constraints, refer to Use Temporal Constraints to Restrict Effective Roles.

`default`

Specifies a default value if the object is created without passing a value. Default values are available for the following data types, and arrays of those types:

- boolean
- number
- object
- string

> **Note**
>
> IDM assumes all default values are valid for the schema.

Default values

You can specify default values in the managed object configuration. If you omit a default value when creating an object, the default value is automatically applied to the object. You can have default values for the following data types, and arrays of those types:

- boolean
- number
- object
- string

For example, the default managed object configuration includes a default value that makes `accountStatus:active`, which effectively replaces the `onCreate` script that was previously used to achieve the same result. The following excerpt from the managed object configuration displays the default value for `accountStatus`:

```
"accountStatus" : {
    "title" : "Status",
    "description" : "Status",
    "viewable" : true,
    "type" : "string",
    "searchable" : true,
    "userEditable" : false,
    "usageDescription" : "",
    "isPersonal" : false,
    "policies" : [
        {
            "policyId": "regexpMatches",
            "params": {
                "regexp": "^(active|inactive)$"
            }
        }
    ],
    "default" : "active"
}
```

> **Note**
>
> IDM assumes all default values are valid for the schema. Although IDM skips policy validation for objects with default values, you can force validation on property values.

Virtual properties

Properties can be derived from other properties within an object. This lets computed and composite values be created in the object. Such derived properties are named *virtual properties*. The value of a virtual property can be calculated in two ways:

- Using a script called by the `onRetrieve` script hook. This script then calculates the current value of the virtual property based on the related properties.
- Using a query to identify the relationship fields to traverse to reach the managed objects whose state is included in the virtual property, and the fields in these managed objects to include in the value of the virtual property.

  These properties are called *relationship-derived virtual properties*.

Virtual properties using `onRetrieve` scripts

The `onRetrieve` script hook lets you run a script when the object is retrieved. In the case of virtual properties, this script gets the data from related properties and uses it to calculate a value for the virtual property. For more information about running scripts on managed objects, refer to Run scripts on managed objects.

Relationship-derived virtual properties

Virtual properties can be calculated by IDM based on relationships and relationship notifications. This means that, rather than calculating the current state when retrieved, the managed object that contains the virtual property is notified of changes in a related object, and the virtual property is recalculated when this notification is received. To configure virtual properties to use relationship notifications, there are two areas that need to be configured:

- The related managed objects must be configured to use relationship notifications. This lets IDM know where to send notifications of changes in related objects.
- To calculate the value of a virtual property, you must configure *which* relationships to check, and in which order, a notification of a change in a related object is received. You configure this using the `queryConfig` property.

The `queryConfig` property tells IDM the sequence of relationship fields it should traverse in order to calculate (or recalculate) a virtual property, and which fields it should return from that related object. This is done using the following fields:

- `referencedRelationshipFields` is an array listing a sequence of relationship fields connecting the current object with the related objects you want to calculate the value of the virtual property from. The first field in the array is a relationship field belonging to the same managed object as the virtual property. The second field is a relationship in the managed object referenced by the first field, and so on.

  For example, the `referencedRelationshipFields` for `effectiveAssignments` is `["roles","assignments"]`. The first field refers to the `roles` relationship field in `managed/user`, which references the `managed/role` object. It then refers to the `assignments` relationship in `managed/role`, which references the `managed/assignment` object. Changes to either related object (`managed/role` or `managed/assignment`) will cause the virtual property value to be recalculated, due to the

`notify`, `notifySelf`, and `notifyRelationships` configurations on managed user, role, and assignment. These configurations ensure that any changes in the relationships between a user and their roles, or their roles, and their assignments, as well as any relevant changes to the roles or assignments themselves, such as the modification of temporal constraints on roles, or attributes on assignments, will be propagated to connected users, so their `effectiveRoles` and `effectiveAssignments` can be recalculated and potentially synced.

- `referencedObjectFields` is an array of object fields that should be returned as part of the virtual property. If this property is not included, the returned properties will be a reference for the related object. To return the entire related object, use `*`.
- `flattenProperties` is a boolean that specifies whether relationship-derived virtual properties should be returned as plain fields rather than as JSON objects with an `_id` and a `_rev`. This property is `false` by default.

  With `flattenProperties` set to `false`, and `referencedObjectFields` set to `name`, the response to a query on a user's `effectiveAssignments` might look something like this:

  ```
  "effectiveAssignments": [
      {
          "name": "MyFirstAssignment",
          "_id": "02b166cc-d7ed-46b7-813f-5ed103145e76",
          "_rev": "2"
      },
      {
          "name": "MySecondAssignment",
          "_id": "7162ddd4-591a-413e-a30b-3a5864bee5ec",
          "_rev": "0"
      }
  ]
  ```

  With `flattenProperties` set to `true`, and `referencedObjectFields` set to `name`, the response to the same query looks like this:

  ```
  "effectiveAssignments": [
      "MyFirstAssignment",
      "MySecondAssignment"
  ]
  ```

  Setting `flattenProperties` to `true` also lets singleton relationship-derived virtual properties be initialized to `null`.

Using `queryConfig`, the virtual property is recalculated when it receives a notice that changes occurred in the related objects. This can be significantly more efficient than recalculating whenever an object is retrieved, while still ensuring the state of the virtual property is correct.

Note

When you change which fields to return using `referencedObjectFields`, changes are not reflected until there is a change in the related object that would trigger the virtual property to be recalculated (as specified by the `notify`, `notifySelf`, and `notifyRelationships` configurations). The calculated state of the virtual property is still correct, but since a change is necessary for the state to be updated, the returned fields will still be based on the previous configuration.

The `effectiveAssignments` property in `managed.json` is an example of a relationship-derived virtual property:

```
"effectiveAssignments" : {
    "type" : "array",
    "title" : "Effective Assignments",
    "description" : "Effective Assignments",
    "viewable" : false,
    "returnByDefault" : true,
    "isVirtual" : true,
    "queryConfig" : {
        "referencedRelationshipFields" : [ "roles", "assignments" ],
        "referencedObjectFields" : [ "*" ]
    },
    "usageDescription" : "",
    "isPersonal" : false,
    "items" : {
        "type" : "object",
        "title" : "Effective Assignments Items"
    }
}
```

Run scripts on managed objects

> **Important**
>
> Before implementing a script, it's highly recommended that you validate the script over REST. Use scripts in a test environment before deploying them to a production environment.

A number of *script hooks* let you manipulate managed objects using scripts. Scripts can be triggered during various stages of the lifecycle of the managed object, and are defined in the managed object schema.

You can trigger scripts when a managed object is created (onCreate), updated (onUpdate), retrieved (onRetrieve), deleted (onDelete), validated (onValidate), or stored in the repository (onStore). You can also trigger a script when a change to a managed object triggers an implicit synchronization operation (onSync).

Post-action scripts let you manipulate objects after they are created (postCreate), updated (postUpdate), and deleted (postDelete).

For more information, refer to:

- Scripting guide
- Script triggers defined in the managed object configuration
- Managed objects reference

Track user metadata

Some self-service features, such as progressive profile completion, privacy and consent, and terms and conditions acceptance, rely on user *metadata* that tracks information related to a managed object state. Such data might include when the object was created, or the date of the most recent change, for example. This metadata is not stored within the object itself, but in a separate resource location.

Because object metadata is stored outside the managed object, state change situations (such as the time of an update) are separate from object changes (the update itself). This separation reduces unnecessary synchronization to targets when the only data that has changed is metadata. Metadata is not returned in a query unless it is specifically requested. Therefore, the volume of data that is retrieved when metadata is not required, is reduced.

To specify which metadata you want to track for an object, add a `meta` stanza to the object definition in your managed object configuration. The following default configuration tracks the `createDate` and `lastChanged` date for managed user objects:

```
{
  "objects" : [
    {
      "name" : "user",
      ...
      "schema" : {
        ...
      },
      "meta" : {
        "property" : "_meta",
        "resourceCollection" : "internal/usermeta",
        "trackedProperties" : [
          "createDate",
          "lastChanged"
        ]
      },
      ...
    },
    ...
  ]
}
```

Important

If you are not using the self-service features that require metadata, you can remove the `meta` stanza from the `user` object in your managed object configuration. Preventing the creation and tracking of metadata where it is not required will improve performance in that scenario.

The metadata configuration includes the following properties:

`property`

The property that will be dynamically added to the managed object schema for this object.

`resourceCollection`

The resource location in which the metadata will be stored.

Adjust your repository to match the location you specify here. It's recommended that you use an `internal` object path and define the storage in your `repo.jdbc.json` or `repo.ds.json` file.

For a JDBC repository, metadata is stored in the `metaobjects` table by default. The `metaobjectproperties` table is used for indexing.

For a DS repository, metadata is stored under `ou=usermeta,ou=internal,dc=openidm,dc=forgerock,dc=com` by default.

User objects stored in a DS repository must include the `ou` specified in the preceding `dnTemplate` attribute. For example:

`trackedProperties`

The properties that will be tracked as metadata for this object. In the previous example, the `createDate` (when the object was created) and the `lastChanged` date (when the object was last modified) are tracked.

You cannot search on metadata and it is not returned in the results of a query unless it is specifically requested. To return all metadata for an object, include `_fields=,_meta/*` in your request. The following example returns a user entry without requesting the metadata:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/bjensen"
{
  "_id": "bjensen",
  "_rev": "000000000444dd1a",
  "mail": "bjensen@example.com",
  "givenName": "Barbara",
  "sn": "Jensen",
  "description": "Created By CSV",
  "userName": "bjensen",
  "telephoneNumber": "1234567",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

The following example returns the same user entry, with their metadata:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/bjensen?_fields=,_meta/*"
{
  "_id": "bjensen",
  "_rev": "000000000444dd1a",
  "mail": "bjensen@example.com",
  "givenName": "Barbara",
  "sn": "Jensen",
  "description": "Created By CSV",
  "userName": "bjensen",
  "telephoneNumber": "1234567",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
  "_meta": {
    "_ref": "internal/usermeta/284273ff-5e50-4fa4-9d30-4a3cf4a5f642",
    "_refResourceCollection": "internal/usermeta",
    "_refResourceId": "284273ff-5e50-4fa4-9d30-4a3cf4a5f642",
    "_refProperties": {
      "_id": "30076e2e-8db5-4b4d-ab91-5351d2da4620",
      "_rev": "000000001ad09f00"
    },
    "createDate": "2018-04-12T19:53:19.004Z",
    "lastChanged": {
      "date": "2018-04-12T19:53:19.004Z"
    },
    "loginCount": 0,
    "_rev": "0000000094605ed9",
    "_id": "284273ff-5e50-4fa4-9d30-4a3cf4a5f642"
  }
}
```

Note

Apart from the `createDate` and `lastChanged` shown previously, the request also returns the `loginCount`. This property is stored by default for all objects, and increments with each login request based on password or social authentication. If the object for which metadata is tracked is not an object that "logs in," this field will remain 0.

The request also returns a `_meta` property that includes relationship information. IDM uses the relationship model to store the metadata. When the `meta` stanza is added to the user object definition, the attribute specified by the `property` (`"property" : "_meta",` in this case) is added to the schema as a uni-directional relationship to the resource collection specified by `resourceCollection`. In this example, the user object's `_meta` field is stored as an `internal/usermeta` object. The `_meta/_ref` property shows the full resource path to the internal object where the metadata for this user is stored.

Users

User objects that are managed by IDM are called *managed users*.

For a JDBC repository, IDM stores managed users in the `managedobjects` table. A second table, `managedobjectproperties`, serves as the index table.

IDM provides RESTful access to managed users, at the context path `/openidm/managed/user`. You can add, change, and delete managed users using the admin UI or over the REST interface. To use the admin UI, select **Manage > User**.

If you are viewing users through the admin UI, the User List page supports specialized filtering with the Advanced Filter option. This lets you build many of the queries shown in Define and call data queries.

Managed users examples

The following examples show how to add, change, and delete users over the REST interface. For a reference of all managed user endpoints and actions, refer to the Managed users endpoint.

You can also use the REST API Explorer as a reference to the managed object REST API.

> **Note**
>
> Some examples in this documentation use client-assigned IDs (such as `bjensen` and `scarter`) when creating objects because it makes the examples easier to read. If you create objects using the admin UI, they are created with server-assigned IDs (such as `55ef0a75-f261-47e9-a72b-f5c61c32d339`). Generally, immutable server-assigned UUIDs are used in production environments.

Retrieve the IDs of all managed users in the repository

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true&_fields=_id"
{
  "result": [
    {
      "_id": "bjensen",
      "_rev": "0000000079b78ace"
    },
    {
      "_id": "scarter",
      "_rev": "0000000070e587a7"
    },
    ...
  ],
  ...
}
```

Query managed users for a specific user

The `_queryFilter` requires double quotes, or the URL-encoded equivalent (`%22`), around the search term. This example uses the URL-encoded equivalent:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=userName+eq+%22scarter%22"
{
  "result": [
    {
      "_id": "scarter",
      "_rev": "0000000070e587a7",
      "userName": "scarter",
      "givenName": "Sam",
      "sn": "Carter",
      "telephoneNumber": "12345678",
      "active": "true",
      "mail": "scarter@example.com",
      "accountStatus": "active",
      "effectiveAssignments": [],
      "effectiveRoles": []
    }
  ],
  ...
}
```

This example uses single quotes around the URL to avoid conflicts with the double quotes around the search term:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/user?_queryFilter=userName+eq+"scarter"'
{
  "result": [
    {
      "_id": "scarter",
      "_rev": "0000000070e587a7",
      "userName": "scarter",
      "givenName": "Sam",
      "sn": "Carter",
      "telephoneNumber": "12345678",
      "active": "true",
      "mail": "scarter@example.com",
      "accountStatus": "active",
      "effectiveAssignments": [],
      "effectiveRoles": []
    }
  ],
  ...
}
```

Retrieve a managed user by their ID

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/scarter"
{
  "_id": "scarter",
  "_rev": "0000000070e587a7",
  "userName": "scarter",
  "givenName": "Sam",
  "sn": "Carter",
  "telephoneNumber": "12345678",
  "active": "true",
  "mail": "scarter@example.com",
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": []
}
```

Add a user with a specific user ID

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "userName": "bjackson",
  "sn": "Jackson",
  "givenName": "Barbara",
  "mail": "bjackson@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd"
}' \
"http://localhost:8080/openidm/managed/user/bjackson"
{
  "_id": "bjackson",
  "_rev": "0000000055c185c5",
  "userName": "bjackson",
  "sn": "Jackson",
  "givenName": "Barbara",
  "mail": "bjackson@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": []
}
```

Add a user with a system-generated ID

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "userName": "pjensen",
  "sn": "Jensen",
  "givenName": "Pam",
  "mail": "pjensen@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd"
}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "_id": "9d92cdc8-8b22-4037-a344-df960ea66194",
  "_rev": "00000000a4bf9006",
  "userName": "pjensen",
  "sn": "Jensen",
  "givenName": "Pam",
  "mail": "pjensen@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": []
}
```

Update a user

This example checks whether user `bjensen` exists, then replaces her telephone number with the new data provided in the request body:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '[
  {
    "operation": "replace",
    "field": "/telephoneNumber",
    "value": "0763483726"
  }
]' \
"http://localhost:8080/openidm/managed/user?_action=patch&_queryFilter=userName+eq+'bjackson'"
{
  "userName": "bjackson",
  "sn": "Jackson",
  "givenName": "Barbara",
  "mail": "bjackson@example.com",
  "telephoneNumber": "0763483726",
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": [],
  "_rev": "000000008c0f8617",
  "_id": "bjackson"
}
```

Delete a user

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/managed/user/bjackson"
{
  "_id": "bjackson",
  "_rev": "000000008c0f8617",
  "userName": "bjackson",
  "sn": "Jackson",
  "givenName": "Barbara",
  "mail": "bjackson@example.com",
  "telephoneNumber": "0763483726",
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": []
}
```

Relationships between objects

Relationships are references between managed objects. Roles and Organizations are implemented using relationships, but you can create relationships between any managed object type.

Create a relationship between two objects

When you have defined a relationship *type*, (such as the `manager` relationship, described in the previous section), you can *reference* one managed user from another, using the `_ref*` relationship properties. Three properties make up a relationship reference:

- `_refResourceCollection` specifies the container of the referenced object (for example, `managed/user`).
- `_refResourceId` specifies the ID of the referenced object. This is generally a system-generated UUID, such as `9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb`. For clarity, this section uses client-assigned IDs such as `bjensen` and `psmith`.
- `_ref` is a derived path that is a combination of `_refResourceCollection` and a URL-encoded `_refResourceId`.

For example, imagine that you are creating a new user, psmith, and that psmith's manager will be bjensen. You would add psmith's user entry, and *reference* bjensen's entry with the `_ref` property, as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "If-None-Match: *" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "sn":"Smith",
  "userName":"psmith",
  "givenName":"Patricia",
  "displayName":"Patti Smith",
  "description" : "psmith - new user",
  "mail" : "psmith@example.com",
  "phoneNumber" : "0831245986",
  "password" : "Passw0rd",
  "manager" : {"_ref" : "managed/user/bjensen"}
}' \
"http://localhost:8080/openidm/managed/user/psmith"
{
  "_id": "psmith",
  "_rev": "00000000ec41097c",
  "sn": "Smith",
  "userName": "psmith",
  "givenName": "Patricia",
  "displayName": "Patti Smith",
  "description": "psmith - new user",
  "mail": "psmith@example.com",
  "phoneNumber": "0831245986",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

Note that relationship information is not returned by default. To show the relationship in psmith's entry, you must explicitly request her manager entry, as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/psmith?_fields=manager"
{
  "_id": "psmith",
  "_rev": "00000000ec41097c",
  "manager": {
    "_ref": "managed/user/bjensen",
    "_refResourceCollection": "managed/user",
    "_refResourceId": "bjensen",
    "_refProperties": {
      "_id": "ffc6f0f3-93db-4939-b9eb-1f8389a59a52",
      "_rev": "0000000081aa991a"
    }
  }
}
```

If a relationship changes, you can query the updated relationship state when any referenced managed objects are queried. So, after creating user psmith with manager bjensen, a query on bjensen's user entry will show a reference to psmith's entry in her `reports` property (because the `reports` property is configured as the `reversePropertyName` of the `manager` property). The following query shows the updated relationship state for bjensen:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/bjensen?_fields=reports"
{
  "_id": "bjensen",
  "_rev": "0000000057b5fe9d",
  "reports": [
    {
      "_ref": "managed/user/psmith",
      "_refResourceCollection": "managed/user",
      "_refResourceId": "psmith",
      "_refProperties": {
        "_id": "ffc6f0f3-93db-4939-b9eb-1f8389a59a52",
        "_rev": "0000000081aa991a"
      }
    }
  ]
}
```

IDM maintains referential integrity by deleting the relationship reference, if the object referred to by that relationship is deleted. In our example, if bjensen's user entry is deleted, the corresponding reference in psmith's `manager` property is removed.

Configure relationship change notification

A relationship exists between two managed objects. By default, when a relationship changes (when it is created, updated, or deleted), the managed objects on either side of the relationship are not *notified* of that change. This means that the *state* of each object with respect to that relationship field is not recalculated until the object is read. This default behavior improves performance, especially in the case where many objects are affected by a single relationship change.

For `roles`, a special kind of relationship, change notification *is* configured by default. The purpose of this default configuration is to notify managed users when any of the relationships that link users, roles, and assignments are manipulated. For more information about relationship change notification in the specific case of managed roles, refer to Roles and relationship change notification.

To change the default configuration, or to set up notification for other relationship changes, use the `notify*` properties in the relationship definition, as described in this section.

A relationship exists between an *origin* object and a *referenced* object. These terms reflect which managed object is specified in the URL (for example `managed/user/psmith`), and which object is referenced by the relationship (`_ref*`) properties. For more information about the relationship properties, refer to Create a relationship between two objects.

In the previous example, a PUT on `managed/user/psmith` with `"manager" : {_ref : "managed/user/bjensen"}`, causes `managed/user/psmith` to be the origin object, and `managed/user/bjensen` to be the referenced object for that relationship, as shown in the following illustration:

Figure 1. Relationship Objects

Note that for the reverse relationship (a PUT on `managed/user/bjensen` with `"reports" : [{_ref = "managed/user/psmith"}]`) `managed/user/bjensen` would be the origin object, and `managed/user/psmith` would be the referenced object.

By default, when a relationship changes, neither the origin object nor the referenced object is *notified* of the change. So, with the PUT on `managed/user/psmith` with `"manager" : {_ref : "managed/user/bjensen"}`, neither psmith's object nor bjensen's object is notified.

> **Note**
>
> Auditing is not tied to relationship change notification and is always triggered when a *relationship* changes. Therefore, relationship changes are audited, regardless of the `notify` and `notifySelf` properties.

To configure relationship change notification, set the `notify` and `notifySelf` properties in your managed object schema. These properties specify whether objects that reference relationships are notified of a relationship change:

`notifySelf`

Notifies the origin object of the relationship change.

In our example, if the `manager` definition includes `"notifySelf" : true`, and if the relationship is changed through a URL that references psmith, then psmith's object would be notified of the change. For example, for a CREATE, UPDATE or DELETE request on the `psmith/manager`, psmith would be notified, but the managed object referenced by this relationship (bjensen) would not be notified.

If the relationship were manipulated through a request to `bjensen/reports`, then bjensen would only be notified if the `reports` relationship specified `"notifySelf" : true`.

notify

Notifies the referenced object of the relationship change. Set this property on the `resourceCollection` of the relationship property.

In our example, assume that the `manager` definition has a `resourceCollection` with a `path` of `managed/user`, and that this object specifies `"notify" : true`. If the relationship changes through a CREATE, UPDATE, or DELETE on the URL `psmith/manager`, then the reference object (`managed/user/bjensen`) would be notified of the change to the relationship.

notifyRelationships

This property controls the propagation of notifications out of a managed object when one of its properties changes through an update or patch, or when that object receives a notification through one of these fields.

The `notifyRelationships` property takes an array of relationships as a value; for example, `"notifyRelationships" : ["relationship1", "relationship2"]`. The relationships specified here are fields defined on the managed object type (which might itself be a relationship).

Notifications are propagated according to the *recipient's* `notifyRelationships` configuration. If a managed object type is notified of a change through one if its relationship fields, the notification is done according to the configuration of the recipient object. To illustrate, look at the `attributes` property in the default `managed/assignment` object:

```
{
    "name" : "assignment",
    "schema" : {
        ...
        "properties" : {
            ...
            "attributes" : {
                "description" : "The attributes operated on by this assignment.",
                "title" : "Assignment Attributes",
                ...
                "notifyRelationships" : ["roles"]
            },
...
```

This configuration means that if an assignment is updated or patched, and the assignment's `attributes` change in some way, all the `roles` connected to that assignment are notified. Because the `role` managed object has `"notifyRelationships" : ["members"]` defined on its `assignments` field, the notification that originated from the change to the assignment attribute is propagated to the connected `roles`, and then out to the `members` of those roles.

So, the `role` is notified through its `assignments` field because an `attribute` in the assignment changed. This notification is propagated out of the `members` field because the role definition has `"notifyRelationships" : ["members"]` on its `assignments` field.

By default, `roles`, `assignments`, and `members` use relationship change notification to ensure that relationship changes are accurately provisioned.

For example, the default `user` object includes a `roles` property with `notifySelf` set to `true`:

```
{
   "name" : "user",
   ...
   "schema" : {
       ...
       "properties" : {
           ...
           "roles" : {
               "description" : "Provisioning Roles",
               ...
               "items" : {
                   "type" : "relationship",
                   ...
                   "reverseRelationship" : true,
                   "reversePropertyName" : "members",
                   "notifySelf" : true,
                   ...
               }
...
```

In this case, `notifySelf` indicates the origin or `user` object. If any changes are made to a relationship referencing a role through a URL that includes a user, the user will be notified of the change. For example, if there is a CREATE on `managed/user/psmith/roles` which specifies a set of references to existing roles, user `psmith` will be notified of the change.

Similarly, the `role` object includes a `members` property. That property includes the following schema definition:

```
{
    "name" : "role",
    ...
    "schema" : {
        ...
        "properties" : {
            ...
            "members" : {
                ...
                "items" : {
                    "type" : "relationship",
                    ...
                    "properties" : {
                        ...
                        "resourceCollection" : [
                            {
                                "notify" : true,
                                "path" : "managed/user",
                                "label" : "User",
                                ...
                            }
                        ]
                    }
...
```

Notice the `"notify" : true` setting on the `resourceCollection`. This setting indicates that if the relationship is created, updated, or deleted through a URL that references that role, all objects in that resource collection (in this case, `managed/user` objects) that are identified as `members` of that role must be notified of the change.

Important

- To notify an object at the end of a relationship that the relationship has changed (using the `notify` property), the relationship *must* be bidirectional (`"reverseRelationship" : true`).
  When an object is notified of a relationship state change (create, delete, or update), part of that notification process involves calculating the changed object state with respect to the changed relationship field. For example, if a managed user is notified that a role has been created, the user object calculates its base state, and the state of its `roles` field, before and after the new role was created. This *before* and *after* state is then reconciled. An object that is referenced by a forward (unidirectional) relationship does not have a field that references that relationship; the object is "pointed-to", but does not "point-back". Because this object cannot calculate its *before* and *after* state with respect to the relationship field, it cannot be notified.
  Similarly, relationships that are notified of changes to the objects that reference them *must* be bidirectional relationships.
  If you configure relationship change notification on a unidirectional relationship, IDM throws an exception.
- You cannot configure relationship change notification in the admin UI; you must update the managed object configuration directly.

Validate relationships between objects

Optionally, you can specify that a relationship between two objects must be validated when the relationship is created. For example, you can indicate that a user cannot reference a role, if that role does not exist.

When you create a new relationship type, validation is disabled by default, because it involves an expensive query to the relationship that is not always required.

To configure validation of a referenced relationship, set `"validate": true` in the managed object configuration. The default schema enables validation for the following relationships:

- For user objects—roles, managers, and reports
- For role objects—members and assignments
- For assignment objects—roles

Create bidirectional relationships

In most cases, you define a relationship between two objects *in both directions*. For example, a relationship between a user and his manager might indicate a *reverse relationship* between the manager and her direct report. Reverse relationships are particularly useful for queries. You might want to query jdoe's user entry to discover who his manager is, *or* query bjensen's user entry to discover all the users who report to bjensen.

You declare a reverse relationship as part of the relationship definition. Consider the following sample excerpt of the default managed object configuration:

```
"reports" : {
    "description" : "Direct Reports",
    "title" : "Direct Reports",
    ...
    "type" : "array",
    "returnByDefault" : false,
    "items" : {
        "type" : "relationship",
        "reverseRelationship" : true,
        "reversePropertyName" : "manager",
        "validate" : true,
        ...
    }
...
```

The `reports` property is a `relationship` between users and managers. So, you can *refer* to a managed user's reports by referencing the `reports`. However, the reports property is also a reverse relationship (`"reverseRelationship" : true`) which means that you can list all users that reference that report.

You can list all users whose `manager` property is set to the currently queried user.

The reverse relationship includes an optional `resourceCollection` that lets you query a set of objects, based on specific fields:

```
"resourceCollection" : [
    {
        "path" : "managed/user",
        "label" : "User",
        "query" : {
            "queryFilter" : "true",
            "fields" : [
                "userName",
                "givenName",
                "sn"
            ]
        }
    }
]
```

The `path` property of the `resourceCollection` points to the set of objects to be queried. If this path is not in the local repository, the link expansion can incur a significant performance cost. Although the `resourceCollection` is optional, the same performance cost is incurred if the property is absent.

The `query` property indicates how you will query this resource collection to configure the relationship. In this case, `"queryFilter" : "true",` indicates that you can search on any of the properties listed in the `fields` array when you are assigning a manager to a user or a new report to a manager.

Grant relationships conditionally

Relationships can be granted dynamically, based on a specified condition. In order to conditionally grant a relationship, the schemas for the resources you are creating a relationship between need to be configured to support conditional association. To do this, three fields in the schema are used:

conditionalAssociation

Boolean. This property is applied to the `resourceCollection` for the grantor of the relationship. For example, the `members` relationship on `managed/role` specifies that there is a conditional association with the `managed/user` resource:

```
"resourceCollection" : [
    {
        "notify" : true,
        "conditionalAssociation" : true,
        "path" : "managed/user",
        "label" : "User",
        "query" : {
            "queryFilter" : "true",
            "fields" : [
                "userName",
                "givenName",
                "sn"
            ]
        }
    }
]
```

conditionalAssociationField

This property is a string, specifying the field used to determine whether a conditional relationship is granted. The field is applied to the `resourceCollection` of the grantee of the relationship. For example, the `roles` relationship on `managed/user` specifies that the conditional association with `managed/role` is defined by the `condition` field in `managed/role`:

```
"resourceCollection" : [
    {
        "path" : "managed/role",
        "label" : "Role",
        "conditionalAssociationField" : "condition",
        "query" : {
            "queryFilter" : "true",
            "fields" : [
                "name"
            ]
        }
    }
]
```

The field name specified will usually be `condition` if you are using default schema, but can be any field that evaluates a condition and has been flagged as `isConditional`.

isConditional

Boolean. This is applied to the field you wish to check to determine whether membership in a relationship is granted. Only one field on a resource can be marked as `isConditional`. For example, in the relationship between `managed/user` and `managed/role`, conditional membership in the relationship is determined by the query filter specified in the `managed/role` `condition` field:

```
"condition" : {
    "description" : "A conditional filter for this role",
    "title" : "Condition",
    "viewable" : false,
    "searchable" : false,
    "isConditional" : true,
    "type" : "string"
}
```

Conditions support both properties and virtual properties derived from other relationships, if the query property has been configured. Conditions are a powerful tool for dynamically creating relationships between two objects. An example of conditional relationships in use is covered in Grant a Role Based on a Condition.

View relationships over REST

By default, information about relationships is not returned as the result of a GET request on a managed object. You must explicitly include the relationship property in the request, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/psmith?_fields=manager"
{
  "_id": "psmith",
  "_rev": "0000000014c0b68d",
  "manager": {
    "_ref": "managed/user/bjensen",
    "_refResourceCollection": "managed/user",
    "_refResourceId": "bjensen",
    "_refProperties": {
      "_id": "42418f09-ad6c-4b77-bf80-2a12d0c44678",
      "_rev": "00000000288b921e"
    }
  }
}
```

To obtain more information about the referenced object (psmith's manager, in this case), you can include additional fields from the referenced object in the query, using the syntax `object/property` (for a simple string value) or `object/*/property` (for an array of values).

The following example returns the email address and contact number for psmith's manager:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/psmith?_fields=manager/mail,manager/telephoneNumber"
{
  "_id": "psmith",
  "_rev": "0000000014c0b68d",
  "manager": {
    "_rev": "000000005bac8c10",
    "_id": "bjensen",
    "telephoneNumber": "12345678",
    "mail": "bjensen@example.com",
    "_ref": "managed/user/bjensen",
    "_refResourceCollection": "managed/user",
    "_refResourceId": "bjensen",
    "_refProperties": {
      "_id": "42418f09-ad6c-4b77-bf80-2a12d0c44678",
      "_rev": "00000000288b921e"
    }
  }
}
```

To query all the relationships associated with a managed object, query the reference (`*_ref`) property of that object. For example, the following query shows all the objects that are referenced by psmith's entry:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/psmith?_fields=*_ref"
{
  "_id": "psmith",
  "_rev": "0000000014c0b68d",
  "reports": [],
  "manager": {
    "_ref": "managed/user/bjensen",
    "_refResourceCollection": "managed/user",
    "_refResourceId": "bjensen",
    "_refProperties": {
      "_id": "42418f09-ad6c-4b77-bf80-2a12d0c44678",
      "_rev": "00000000288b921e"
    }
  },
  "roles": [],
  "_meta": {
    "_ref": "internal/usermeta/601a3086-8c64-4966-b33c-7a213b13d859",
    "_refResourceCollection": "internal/usermeta",
    "_refResourceId": "601a3086-8c64-4966-b33c-7a213b13d859",
    "_refProperties": {
      "_id": "9de71bd7-1e1b-462e-b565-ac0a7d2f9269",
      "_rev": "0000000037f79a00"
    }
  },
  "authzRoles": [],
  "_notifications": [
    {
      "_ref": "internal/notification/3000bb64-4619-490a-8c4b-50ae7ca6b20c",
      "_refResourceCollection": "internal/notification",
      "_refResourceId": "3000bb64-4619-490a-8c4b-50ae7ca6b20c",
      "_refProperties": {
        "_id": "f54b6f84-7d3f-4486-a7c1-676fca03eeab",
        "_rev": "00000000748da107"
      }
    }
  ]
}
```

To expand that query to show all fields within each relationship, add a wildcard as follows:

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

Which outputs the following:

```
{
  "_id": "psmith",
  "_rev": "0000000014c0b68d",
  "reports": [],
  "manager": {
    "_rev": "000000005bac8c10",
    "_id": "bjensen",
    "userName": "bjensen",
    "givenName": "Babs",
    "sn": "Jensen",
    "telephoneNumber": "12345678",
    "active": "true",
    "mail": "bjensen@example.com",
    "accountStatus": "active",
    "effectiveAssignments": [],
    "effectiveRoles": [],
    "_ref": "managed/user/bjensen",
    "_refResourceCollection": "managed/user",
    "_refResourceId": "bjensen",
    "_refProperties": {
      "_id": "42418f09-ad6c-4b77-bf80-2a12d0c44678",
      "_rev": "00000000288b921e"
    }
  },
  "roles": [],
  "_meta": {
    "_rev": "0000000079e86d8d",
    "_id": "601a3086-8c64-4966-b33c-7a213b13d859",
    "createDate": "2020-07-29T08:52:20.061794Z",
    "lastChanged": {
      "date": "2020-07-29T11:52:16.424167Z"
    },
    "loginCount": 0,
    "_ref": "internal/usermeta/601a3086-8c64-4966-b33c-7a213b13d859",
    "_refResourceCollection": "internal/usermeta",
    "_refResourceId": "601a3086-8c64-4966-b33c-7a213b13d859",
    "_refProperties": {
      "_id": "9de71bd7-1e1b-462e-b565-ac0a7d2f9269",
      "_rev": "0000000037f79a00"
    }
  },
  "authzRoles": [],
  "_notifications": [
    {
      "_rev": "00000000d93a6598",
      "_id": "3000bb64-4619-490a-8c4b-50ae7ca6b20c",
      "notificationType": "info",
      "message": "Your profile has been updated.",
      "createDate": "2020-07-29T11:52:16.517200Z",
      "_ref": "internal/notification/3000bb64-4619-490a-8c4b-50ae7ca6b20c",
      "_refResourceCollection": "internal/notification",
      "_refResourceId": "3000bb64-4619-490a-8c4b-50ae7ca6b20c",
      "_refProperties": {
        "_id": "f54b6f84-7d3f-4486-a7c1-676fca03eeab",
        "_rev": "00000000748da107"
      }
    }
  ]
}
```

> **Note**
>
> Metadata is implemented using the relationships mechanism so when you request all relationships for a user (with `_ref/`), you will also get all the metadata for that user, if metadata is being tracked. For more information, refer to Track user metadata.

View relationships in graph form

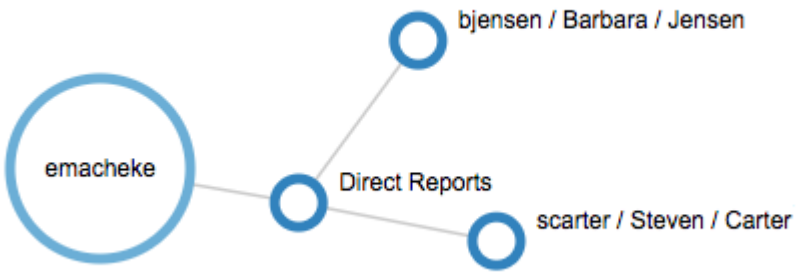
The *Identity Relationships widget* gives a visual display of the relationships between objects.

Add the Identity Relationships widget to a dashboard

This widget is not displayed on any dashboard by default. You can add it as follows:

1. Log in to the admin UI.
2. From the navigation bar, click **Dashboards**, and select a dashboard. Alternatively, create a dashboard.
3. On the applicable dashboard page, click **Add Widget**.
4. In the **Add Widget** window, click the drop-down menu, scroll down to the **Utilities** item, and select **Identity Relationships**.
5. Click **Settings**, and make selections from the following drop-down menus:
   - **Widget Size**—Select from **Small**, **Medium**, or **Large**.
   - **Chart Type**:
     - **Collapsible Tree Layout**:

     - **Radial Layout**:



   - **Default Object**—Select the object for which to display relationships; for example, **User**.
   - **Display/Search Property**—Select one or more properties to search on the default object that will be displayed in the widget; for example, **userName** and **city**.

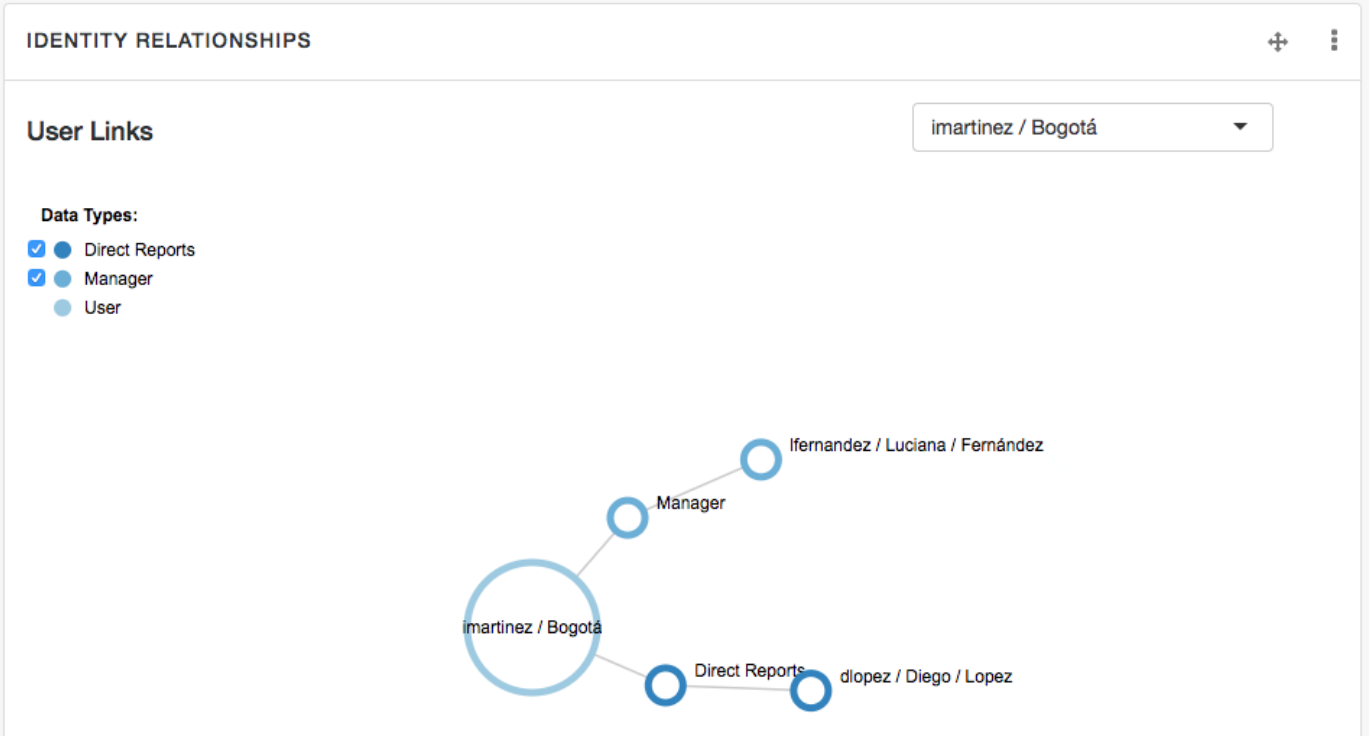
6. Optionally, click **Preview** to view a widget preview.
7. Click **Add** to add the widget to the dashboard.

Use the Identity Relationships widget

To use the **Identity Relationships** widget:

1. Select a user within the widget.

   The following example shows relationships for **imartinez**, including **Manager** and **Direct Reports**:



2. You can interact with the graph in multiple ways:
   - Select or clear **Data Types** to filter information.
   - Click and drag the graph for a different view.
   - Double-click a user to view their profile.

Manage relationships using the admin UI

This section describes how to set up relationships between managed objects by using the admin UI. You can set up a relationship between any object types. The examples in this section demonstrate how to set up a relationship between users and devices, such as IoT devices.

For illustration purposes, these examples assume that you have started IDM and already have some managed users. If this is not the case, start the server with the sample configuration described in Synchronize data from a CSV file to IDM, and run a reconciliation to populate the managed user repository.

In the following procedures, you will:

- Create a new managed object type named *Device* and add a few devices, each with unique serial numbers (see Create a New Device Object Type).
- Set up a bidirectional relationship between the Device object and the managed User object (see Configure the Relationship Between a Device and a User).
- Demonstrate the relationships, assign devices to users, and show relationship validation (see Demonstrate the Relationship).

Create a new Device object type

This procedure illustrates how to set up a new *Device* managed object type, and add properties to collect information such as model, manufacturer, and serial number for each device.

1. From the navigation bar, click **Configure > Managed Objects**.
2. On the **Managed Objects** page, click **New Managed Object**.
3. On the **New Managed Object** page, enter information in the following fields, and click **Save**:

Field	Value
Managed Object Name	`Device`
Readable Title	`Device`
Managed Object Icon	`fa-mobile-phone`
Material Design Icon	`phone`
Description	`Devices`

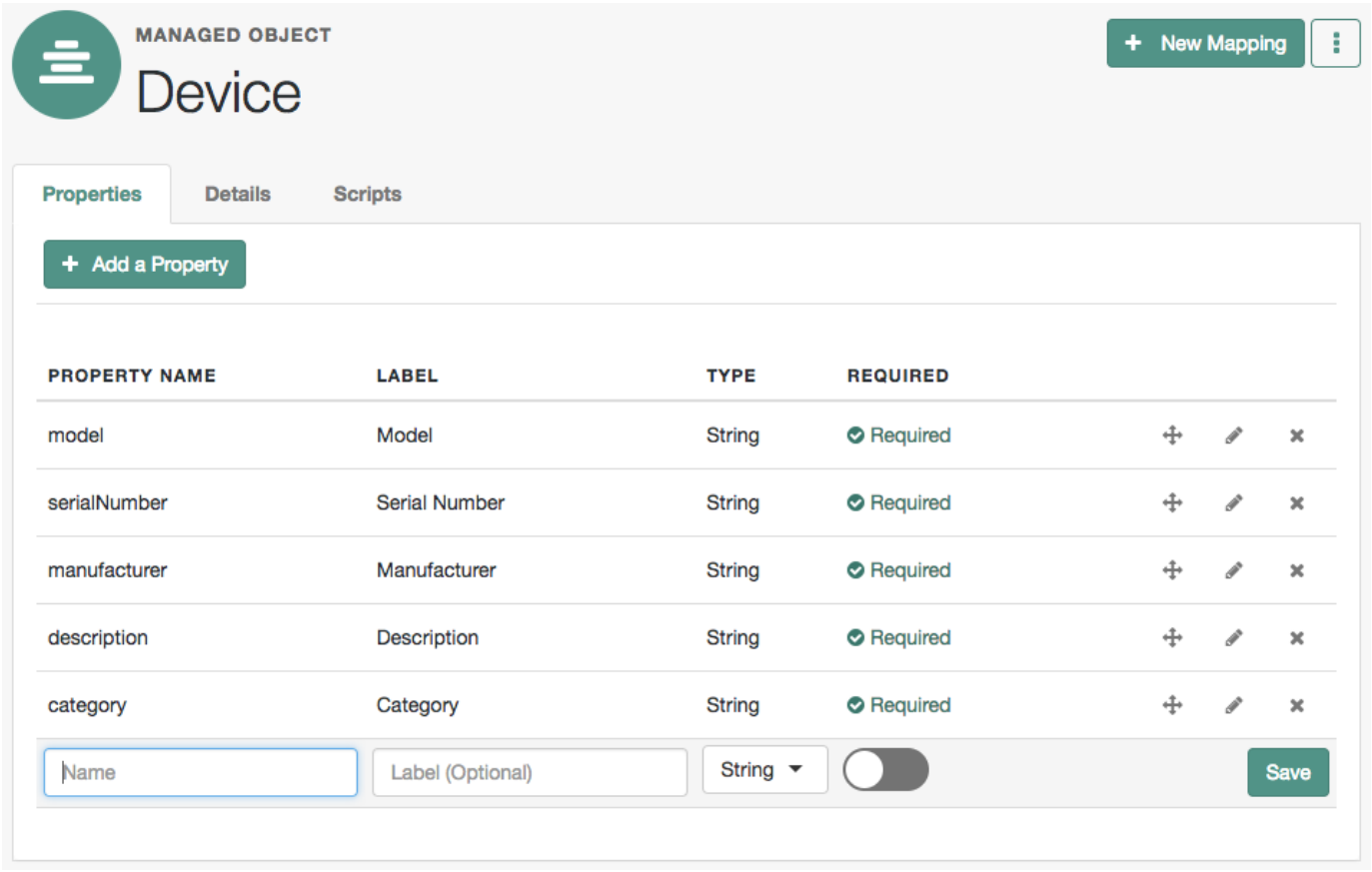
   The **Managed Objects > Device** page displays.

4. Click the **Properties** tab.
5. For each following property, click **Add a Property**, enter the information, and click **Save**:

Property Name	Label	Type	Required
model	Model	String	
serialNumber	Serial Number	String	
manufacturer	Manufacturer	String	
description	Description	String	

Property Name	Label	Type	Required
category	Category	String	✓

After you finish, the properties list should look like this:



6. From the navigation bar, click **Manage** > **Device**.
7. For each of the following devices, on the **Device List** page, click **New Device**, enter the applicable information, and click **Save**:

Device 1

Field	Value
Model	`Generic Phone`
Serial Number	`Phone-1`
Manufacturer	`PhoneCo`
Description	`Entry level phone`
Category	`Smart Phone`

Device 2

Field	Value
Model	`Generic Watch`
Serial Number	`Watch-1`
Manufacturer	`WatchCo`
Description	`Entry level watch`
Category	`Smart Watch`

Device 3

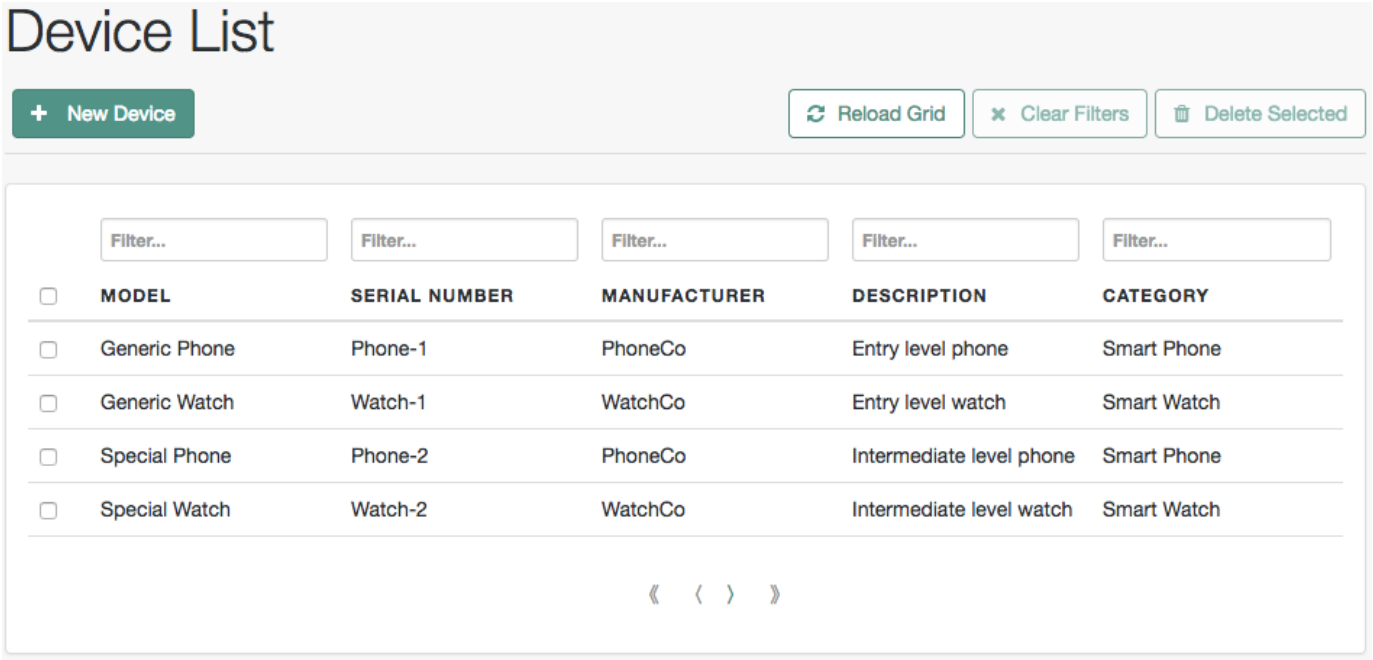
Field	Value
Model	`Special Phone`
Serial Number	`Phone-2`
Manufacturer	`PhoneCo`
Description	`Intermediate level phone`
Category	`Smart Phone`

Device 4

Field	Value
Model	`Special Watch`
Serial Number	`Watch-2`
Manufacturer	`WatchCo`
Description	`Intermediate level watch`
Category	`Smart Watch`

8. From the navigation bar, click **Manage** > **Device**.

   The **Device List** page should look similar to the following:



Note

The other procedures in this topic assume that you have added these devices.

Configure the relationship between a device and a user

To set up a relationship between the Device object type and the User object type, you must identify the specific property on each object that will form the basis of the relationship. For example, a device must have an *owner* and a user can own one or more *devices*. The property *type* for each of these must be *relationship*.

In this procedure, you will update the managed Device object type to add a new Relationship type property named `owner`. You will then link that property to a new property on the managed User object, named `device`. At the end of the procedure, the updated object types will look as follows:

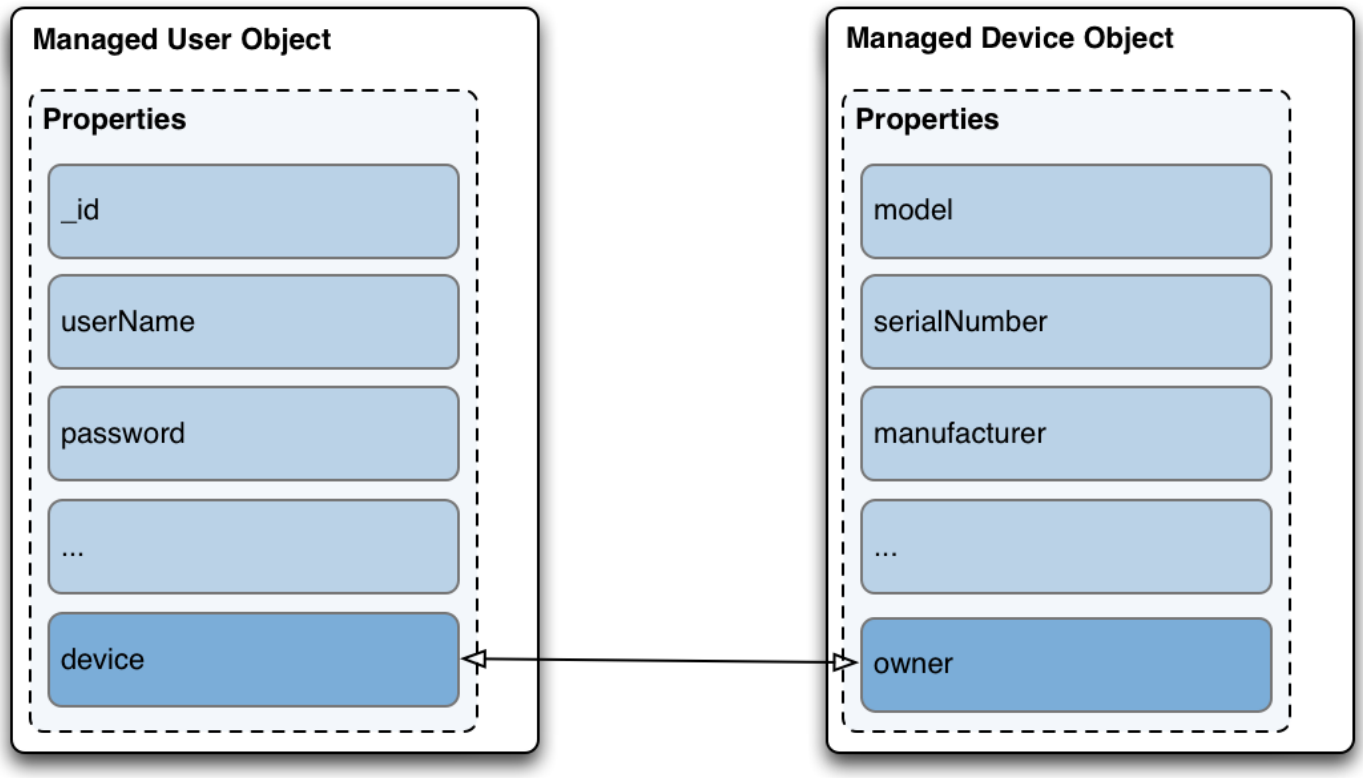


Figure 1. Relationship Properties on User and Device Objects

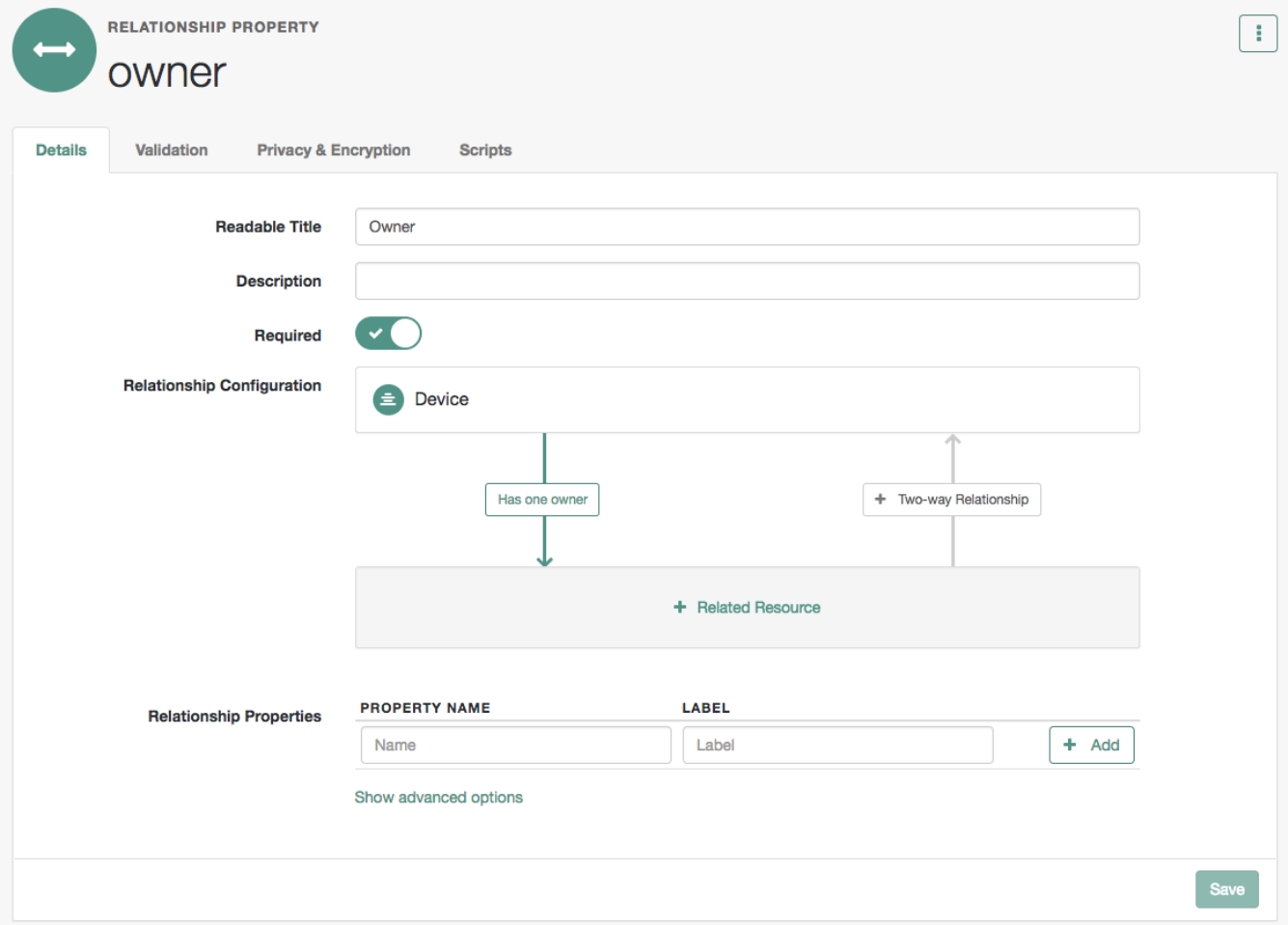
1. Create a new relationship property on the Device object:
    - From the navigation bar, click **Configure > Managed Objects**, and select the **Device** object.
    - On the **Managed Objects > Device** page, click the **Properties** tab.
2. Click **Add a Property**, enter the information, and click **Save**

Property Name	Label	Type	Required
owner	Owner	Relationship	

> **Note**
>
> You cannot change the property **Type** after creation. If you create a property with an incorrect **Type**, you must delete the property and recreate it.

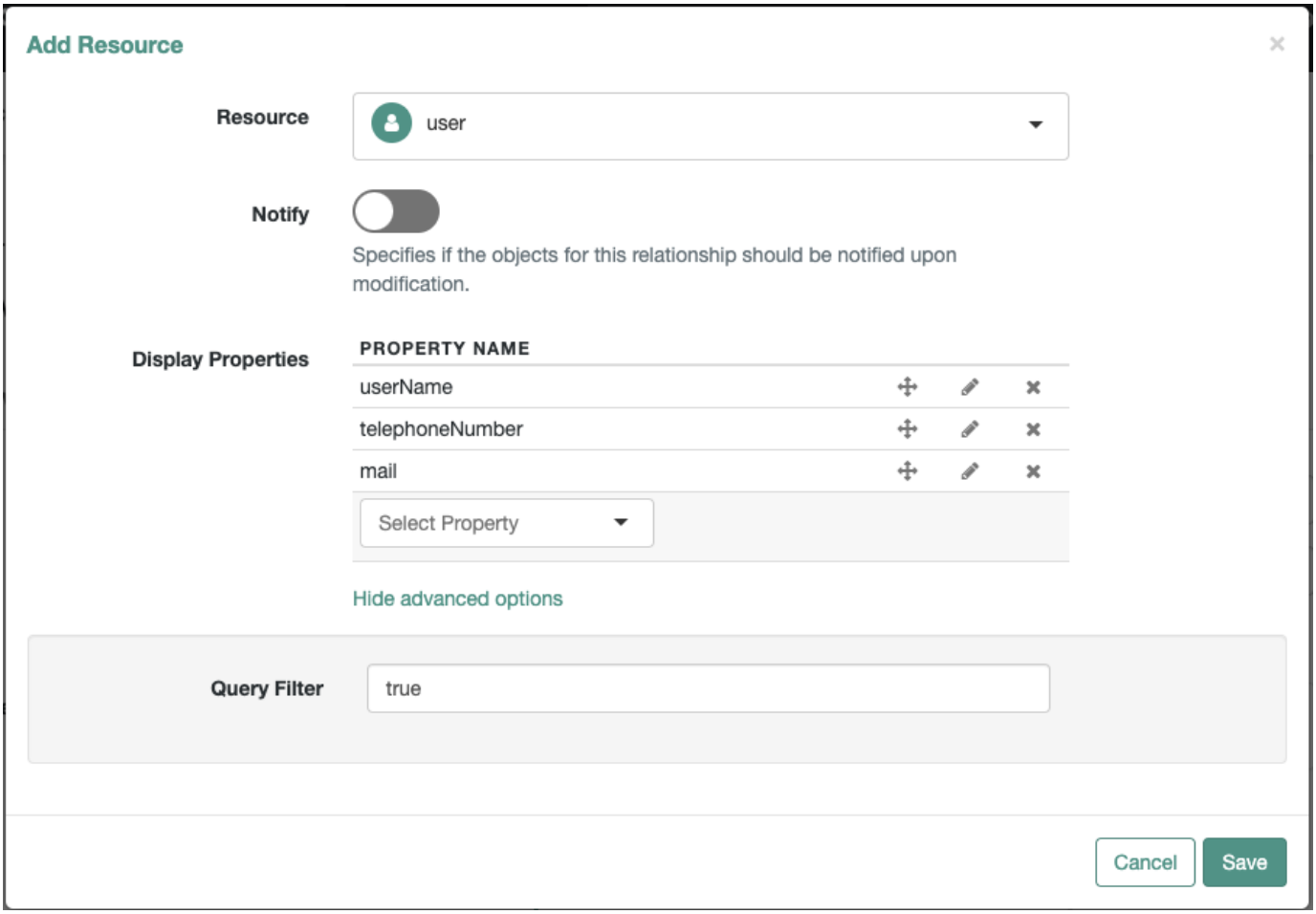
3. Click the **owner** property row.

   The **Details** tab displays the current **Relationship Configuration**:

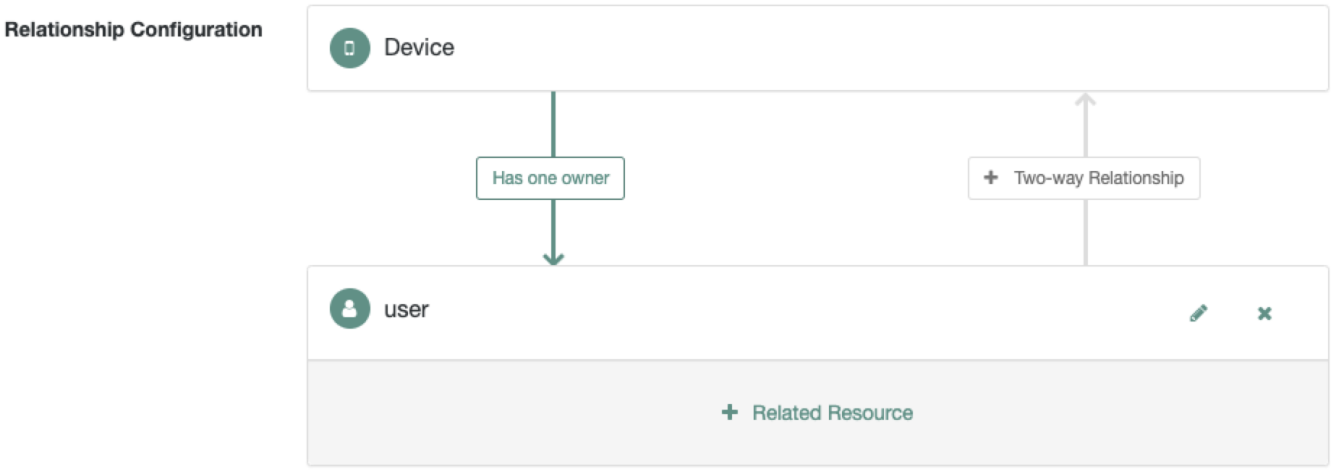


4. Click the **+ Related Resource** area.
5. In the **Add Resource** window, select **user** from the **Resource** drop-down list.

   This sets up a relationship between the Device object and the managed user object.
6. From the **Display Properties** drop-down list, select the user object properties to display when viewing a user's devices in the UI. For example, you may want to access a user's **userName**, **mail**, and **telephoneNumber**.
7. Click **Show advanced options**. Notice that the **Query Filter** field is set to **true**. This setting lets you search on any selected **Display Properties** when assigning a device to a user.
8. Click **Save**.

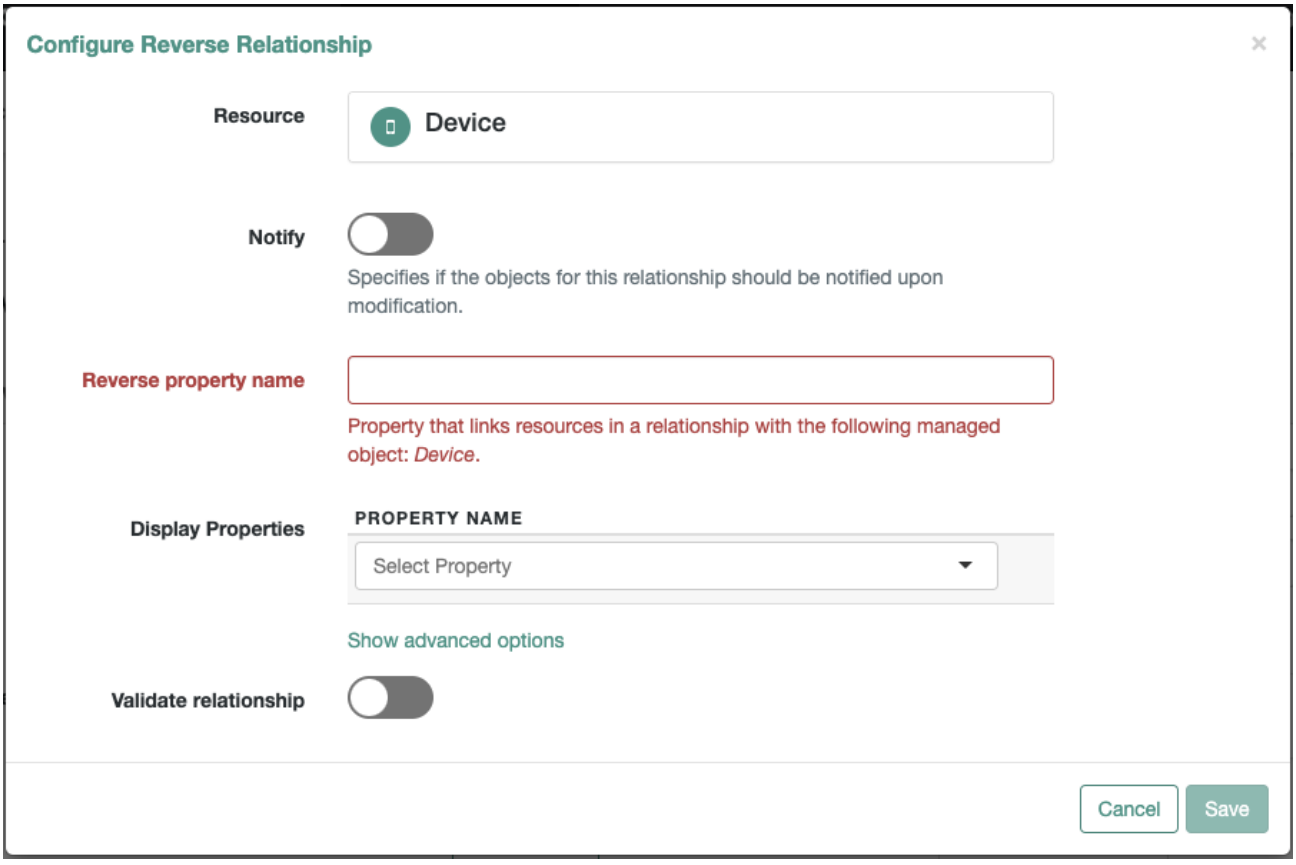


The **Managed** > **Device** > **owner** page now displays the one-way relationship between a device and a user.



9. Click **Save**.
10. To configure the reverse relationship, click **+ Two-way Relationship**:
    1. In the **Reverse Relationship** pop-up, select **Has Many**. This indicates a single user can have more than one device.

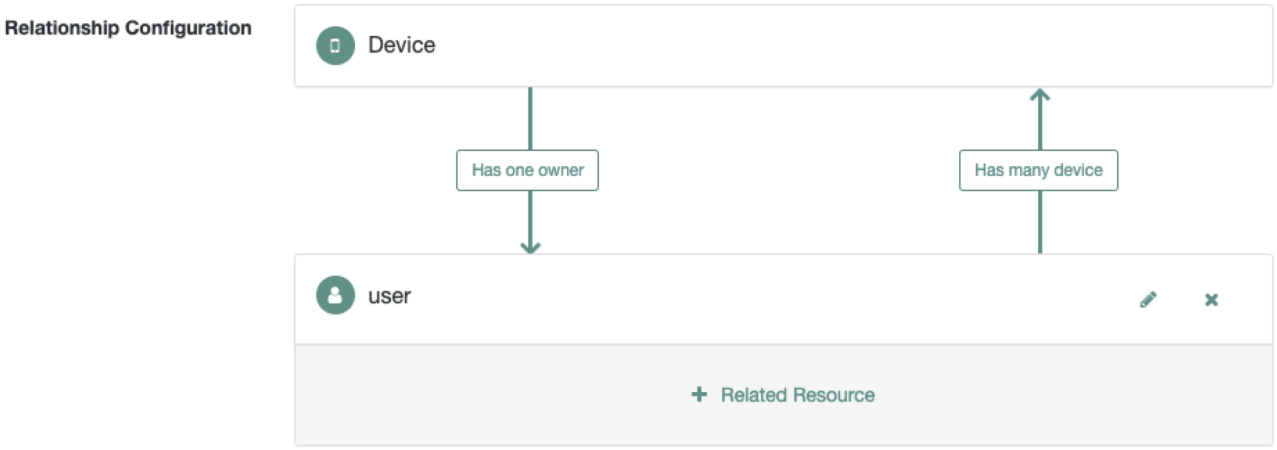
       The **Configure Reverse Relationship** window displays.



2. In the **Reverse property name** field, enter the new property name that will be created in the managed user object type, **device** for this example.
3. From the **Display Properties** drop-down list, select the properties of the device object to display when viewing a user in the UI. For example, you might want to access the **model** and **serialNumber** of each device.
4. Click **Show advanced options**. Notice that the **Query Filter** field is set to **true**. This setting allows you to search on any of the selected **Display Properties** when assigning a device to a user.
5. Enable **Validate relationship**.

   This setting ensures the relationship is valid when a device is assigned to a user. IDM verifies the user and device objects exist, and that the device has not already been assigned to another user.
6. Click **Save**.

   The **Managed > Device > owner** page now displays the two-way relationship showing that a user objects can have many devices.



7. Click **Save**.

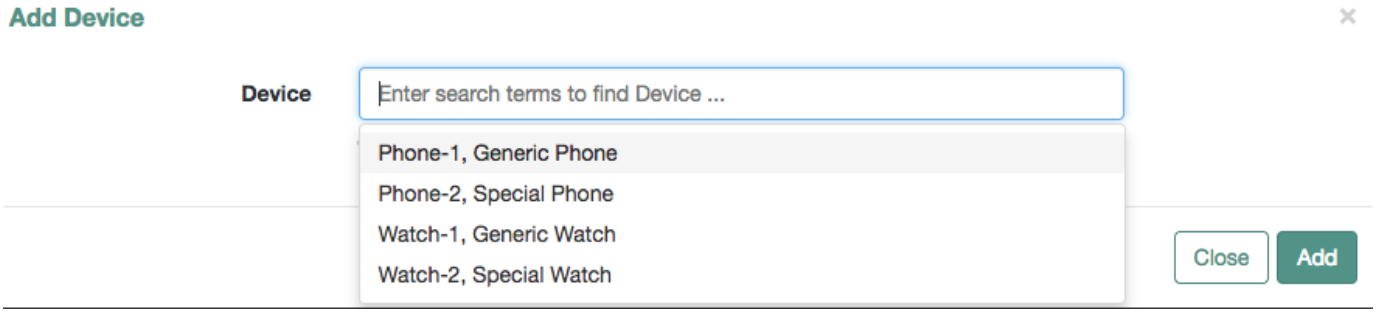
11. From the navigation bar, click **Configure > Managed Objects**.
12. On the **Managed Objects** page, click **User**.
13. On The **Managed Objects > user** page, click the **Properties** tab.

    Notice the **device** property was created automatically when you configured the relationship.

Demonstrate the relationship

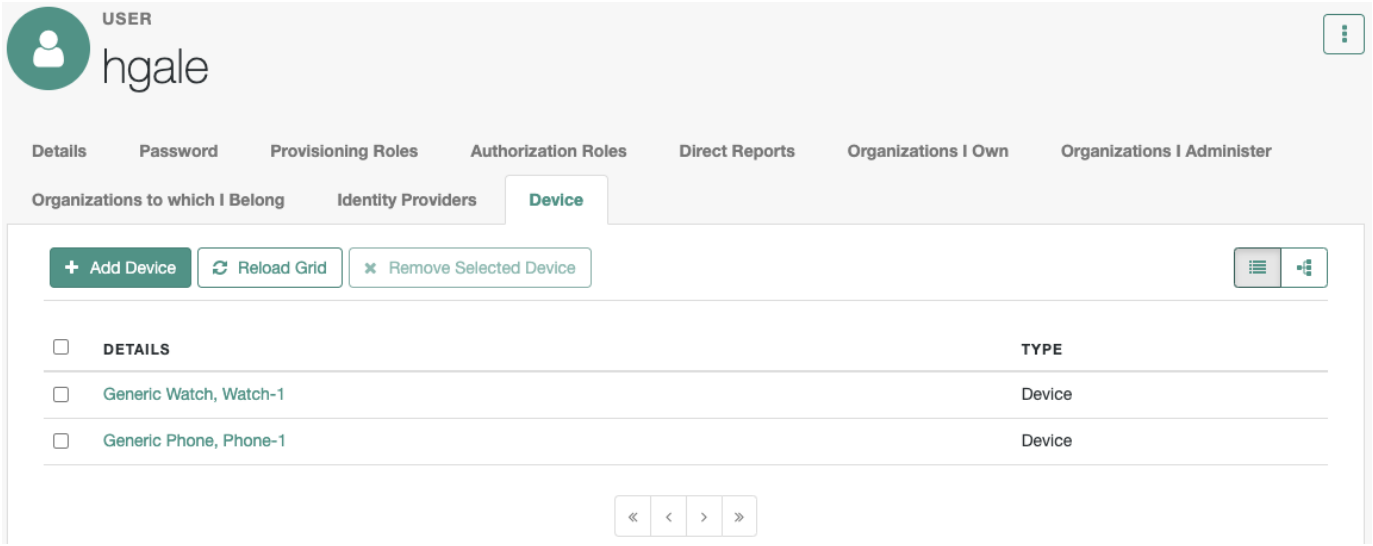
This procedure demonstrates how devices can be assigned to users, based on the relationship configuration that you set up in the previous procedures.

1. From the navigation bar, click **Manage > User**.
2. On the **User List** page, click a user entry.
3. On the **User > userName** page, click the **Device** tab, and then click **Add Device**.
4. In the **Add Device** window, click the **Device** field to display the list of devices that you added in the previous procedure.



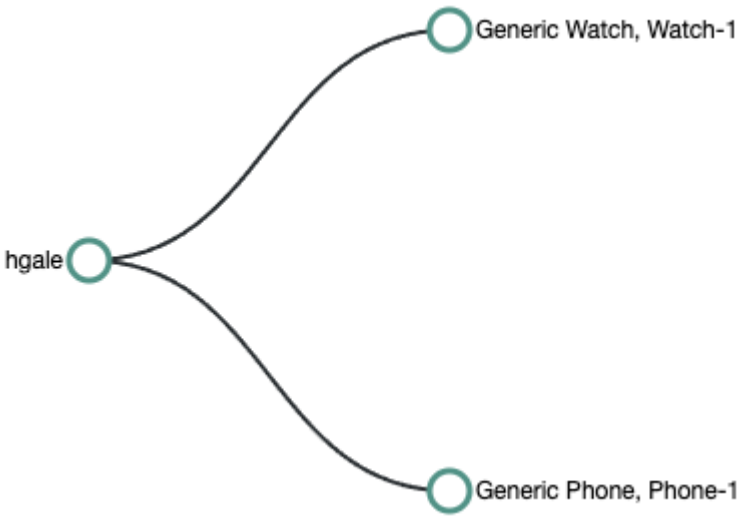
5. Select two devices, and click **Add**.

   The **Device** tab displays the added devices.



6. Click the **Show Chart** button.

   A graphical representation of the relationship between the user and her devices is displayed:



7. You can also assign an owner to a device.

   From the navigation bar, click **Manage > Device**, and click a device that you did not assign in the previous step.

8. On the **Device > model** page, click **Add Owner**.

9. In the **Add Owner** window, select a user, and click **Add**.

10. Click **Save**.

> **Tip**
>
> To demonstrate the relationship validation, try to assign a device that has already been assigned to a different user. The UI displays the error message **Conflict with Existing Relationship**.

View the relationship configuration in the UI

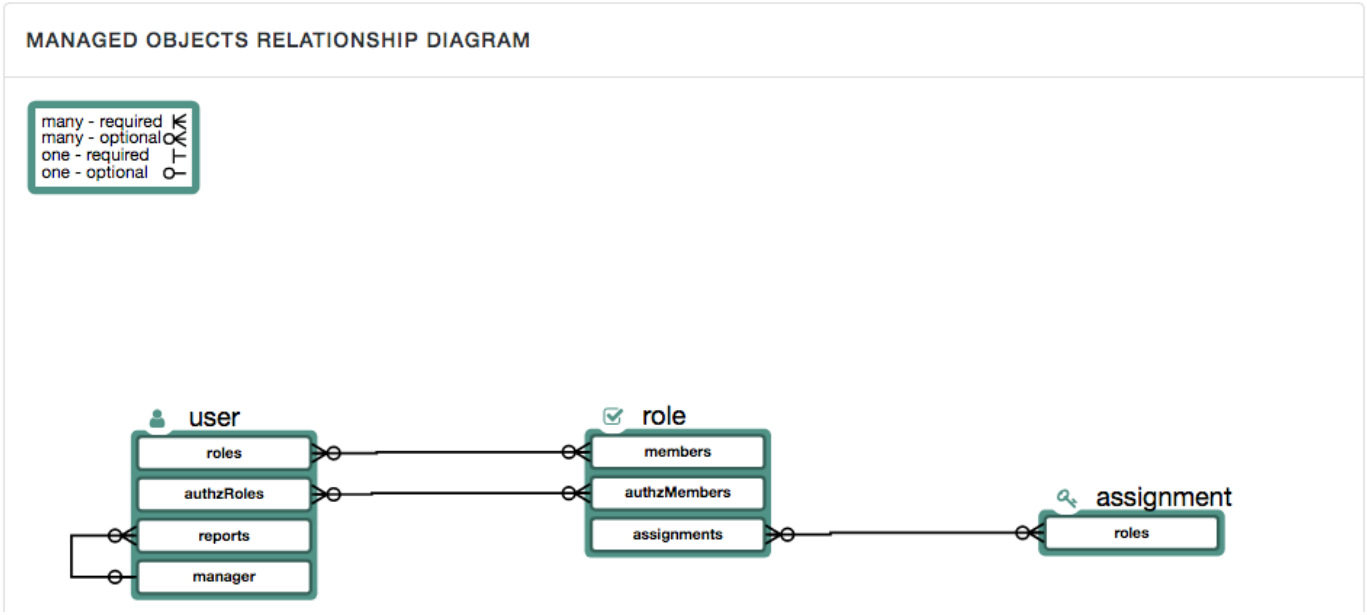
The *Managed Objects Relationship Diagram* provides a visual display of the relationship configuration between managed objects. Unlike the Identity Relationships widget, described in View relationships in graph form, this widget does not show the actual relationship data, but rather shows the configured relationship types.

This widget is not displayed on any dashboard by default. You can add it as follows:

1. Log in to the admin UI.
2. From the navigation bar, click **Dashboards**, and select a dashboard. Alternatively, create a dashboard.
3. On the applicable dashboard page, click **Add Widget**.
4. In the **Add Widget** window, click the drop-down menu, scroll down to the **Utilities** item, and select **Managed Objects Relationship Diagram**.

> **Note**
>
> There are no configurable settings for this widget.

5. The **Preview** button shows the current relationship configuration. The following image shows the relationship configuration for a basic IDM installation with no specific configuration:



The legend indicates which relationships are required, optional, one-to-one, and one-to-many.

Roles

The managed *role* object is a default managed object type that uses the relationships mechanism. You should understand how relationships work before you read about IDM roles.

IDM role types

IDM supports two types of roles:

- *Provisioning roles* : used to specify how objects are provisioned to an external system.

  Provisioning roles are created as managed roles, at the context path `openidm/managed/role/role-name`, and are granted to managed users as values of the user's `roles` property.

- *Authorization roles* : used to specify the authorization rights of a managed object internally, within IDM.

  Authorization roles are created as internal roles, at the context path `openidm/internal/role/role-name`, and are granted to managed users as values of the user's `authzRoles` property.

Provisioning roles and authorization roles use relationships to link the role to the managed object to which it applies. Authorization roles can also be granted statically, during authentication, with the `defaultUserRoles` property.

Managed roles

> **Note**
>
> For information about internal authorization roles, and how IDM controls authorization to its own endpoints, refer to Authorization and roles.

Managed roles are defined like any other managed object, and are granted to users through the *relationships* mechanism. A managed role can be granted manually, as a static value of the user's `roles` attribute, or dynamically, as a result of a condition or script. For example, a user might be granted a role such as `sales-role` dynamically, if that user is in the `sales` organization.

A user's `roles` attribute takes an array of *references* as a value, where the references point to the managed roles. For example, if user bjensen has been granted two roles (`employee` and `supervisor`), the value of bjensen's `roles` attribute would look something like the following:

```
"roles": [
  {
    "_ref": "managed/role/employee",
    "_refResourceCollection": "managed/role",
    "_refResourceId": "employee",
    "_refProperties": {
      "_grantType": "",
      "_id": "bb399428-21a9-4b01-8b74-46a7ac43e0be",
      "_rev": "00000000e43e9ba7"
    }
  },
  {
    "_ref": "managed/role/supervisor",
    "_refResourceCollection": "managed/role",
    "_refResourceId": "supervisor",
    "_refProperties": {
      "_grantType": "",
      "_id": "9f7d124b-c7b1-4bcf-9ece-db4900e37c31",
      "_rev": "00000000e9c19d26"
    }
  }
]
```

The `_refResourceCollection` is the container that holds the role. The `_refResourceId` is the ID of the role. The `_ref` property is a resource path that is derived from the `_refResourceCollection` and the URL-encoded `_refResourceId`. `_refProperties` provides more information about the relationship.

Important

Some of the examples in this documentation set use client-assigned IDs (such as `bjensen` and `scarter`) for the user objects because it makes the examples easier to read. If you create objects using the admin UI, they are created with server-assigned IDs (such as `55ef0a75-f261-47e9-a72b-f5c61c32d339`). This particular example uses a client-assigned role ID that is the same as the role name. All other examples in this chapter use server-assigned IDs. Generally, immutable server-assigned UUIDs are used for all managed objects in production environments.

Manipulate roles

These sections show the REST calls to create, read, update, and delete managed roles, and to grant roles to users. For information about using roles to provision users to external systems, refer to Use assignments to provision users.

Create a role

Use the admin UI

1. From the navigation bar, click **Manage** > **Role**.
2. On the **Roles** page, click **New Role**.
3. On the **New Role** page, enter a name and description, and click **Save**.

4. Optionally, do any of the following, and click **Save**:
    - To restrict the role grant to a set time period, enable **Temporal Constraint**, and set the **Timezone Offset**, **Start Date**, and **End Date**.
    - To define a query filter that dynamically grants the role to members, enable **Condition**, and define the query.

For more information, refer to Use temporal constraints to restrict effective roles and Grant Roles Dynamically.

Use REST

To create a role, send a PUT or POST request to the `/openidm/managed/role` context path. The following example creates a managed role named `employee`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "name": "employee",
  "description": "Role granted to workers on the company payroll"
}' \
"http://localhost:8080/openidm/managed/role?_action=create"
{
  "_id": "5790220a-719b-49ad-96a6-6571e63cbaf1",
  "_rev": "0000000079c6644f",
  "name": "employee",
  "description": "Role granted to workers on the company payroll"
}
```

> **Note**
>
> By default, the role `name` must be unique. To change this behavior, adjust the policy validation on the `role` property in your managed object configuration.

This `employee` role has no corresponding *assignments*. Assignments are what enables the provisioning logic to the external system. Assignments are created and maintained as separate managed objects, and are referred to within role definitions. For more information about assignments, refer to Use assignments to provision users.

List roles

To list all managed roles over REST, query the `openidm/managed/role` endpoint. The following example shows the `employee` role that you created in the previous example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/role?_queryFilter=true"
{
  "result": [
    {
      "_id": "5790220a-719b-49ad-96a6-6571e63cbaf1",
      "_rev": "0000000079c6644f",
      "name": "employee",
      "description": "Role granted to workers on the company payroll"
    }
  ],
  ...
}
```

To display all configured managed roles in the admin UI, select **Manage > Role**.

If you have a large number of roles, select Advanced Filter to build a more complex query filter to display only the roles you want.

Grant roles to a user

You grant roles to users through the relationship mechanism. Relationships are essentially references from one managed object to another; in this case, from a user object to a role object. For more information about relationships, refer to Relationships between objects.

You can grant roles *statically* or *dynamically*.

To grant a role statically, you must do one of the following:

- Update the value of the user's `roles` property to reference the role.
- Update the value of the role's `members` property to reference the user.

Dynamic role grants use the result of a condition or script to update a user's list of roles.

Grant roles statically

Grant a role to a user statically using the REST interface or the admin UI as follows:

Use REST

Use one of the following methods to grant a role to a user over REST:

- Add the user as a role member. The following example adds user scarter as a member of the role (`5790220a-719b-49ad-96a6-6571e63cbaf1`):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "_ref":"managed/user/scarter",
  "_refProperties":{}
}' \
"http://localhost:8080/openidm/managed/role/5790220a-719b-49ad-96a6-6571e63cbaf1/members?
_action=create"
{
  "_id": "4c32ae53-abed-45f8-bc84-c367e2b0e194",
  "_rev": "00000000c67a99ce",
  "_ref": "managed/user/scarter",
  "_refResourceCollection": "managed/user",
  "_refResourceId": "scarter",
  "_refProperties": {
    "_id": "4c32ae53-abed-45f8-bc84-c367e2b0e194",
    "_rev": "00000000c67a99ce"
  }
}
```

Note

This preferred method does not incur an unnecessary performance cost when working with a role that contains many members.

- Update the user's `roles` property to refer to the role.

The following example grants the `employee` role (`5790220a-719b-49ad-96a6-6571e63cbaf1`) to user scarter:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/roles/-",
    "value": {"_ref" : "managed/role/5790220a-719b-49ad-96a6-6571e63cbaf1"}
  }
]' \
"http://localhost:8080/openidm/managed/user/scarter"
{
  "_id": "scarter",
  "_rev": "000000003be825ce",
  "mail": "scarter@example.com",
  "givenName": "Steven",
  "sn": "Carter",
  "description": "Created By CSV",
  "userName": "scarter",
  "telephoneNumber": "1234567",
  "accountStatus": "active",
  "effectiveRoles": [
    {
      "_ref": "managed/role/5790220a-719b-49ad-96a6-6571e63cbaf1"
    }
  ],
  "effectiveAssignments": []
}
```

Note that scarter's `effectiveRoles` attribute has been updated with a reference to the new role. For more information about effective roles and effective assignments, refer to Effective roles and effective assignments.

When you update a user's existing roles array, use the `-` special index to add the new value to the set. For more information, refer to *Set semantic arrays* in Patch Operation: Add.

- Update the role's `members` property to refer to the user.

  The following sample command makes scarter a member of the `employee` role:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/members/-",
    "value": {"_ref" : "managed/user/scarter"}
  }
]' \
"http://localhost:8080/openidm/managed/role/5790220a-719b-49ad-96a6-6571e63cbaf1"
{
  "_id": "5790220a-719b-49ad-96a6-6571e63cbaf1",
  "_rev": "0000000079c6644f",
  "name": "employee",
  "description": "Role granted to workers on the company payroll"
}
```

The `members` property of a role is not returned by default in the output. To show all members of a role, you must specifically request the relationship properties (`*_ref`) in your query. The following example lists the members of the `employee` role (currently only scarter):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/role/5790220a-719b-49ad-96a6-6571e63cbaf1?_fields=*_ref,name"
{
  "_id": "5790220a-719b-49ad-96a6-6571e63cbaf1",
  "_rev": "0000000079c6644f",
  "name": "employee",
  "assignments": [],
  "members": [
    {
      "_ref": "managed/user/scarter",
      "_refResourceCollection": "managed/user",
      "_refResourceId": "scarter",
      "_refProperties": {
        "_id": "7ad15a7b-6806-487b-900d-db569927f56d",
        "_rev": "0000000075e09cbf"
      }
    }
  ]
}
```

- You can replace an existing role grant with a new one by using the `replace` operation in your patch request.

The following command replaces scarter's entire `roles` entry (that is, overwrites any existing roles) with a single entry, the reference to the `employee` role (ID `5790220a-719b-49ad-96a6-6571e63cbaf1`):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "replace",
    "field": "/roles",
    "value": [
      {"_ref":"managed/role/5790220a-719b-49ad-96a6-6571e63cbaf1"}
    ]
  }
]' \
"http://localhost:8080/openidm/managed/user/scarter"
{
  "_id": "scarter",
  "_rev": "00000000da112702",
  "mail": "scarter@example.com",
  "givenName": "Steven",
  "sn": "Carter",
  "description": "Created By CSV",
  "userName": "scarter",
  "telephoneNumber": "1234567",
  "accountStatus": "active",
  "effectiveRoles": [
    {
      "_ref": "managed/role/5790220a-719b-49ad-96a6-6571e63cbaf1"
    }
  ],
  "effectiveAssignments": []
}
```

Use the admin UI

Use one of the following UI methods to grant a role to a user:

Update the user entry:

1. Select **Manage > User**, and select the user to whom you want to grant the role.
2. Select the **Provisioning Roles** tab, and select **Add Provisioning Roles**.
3. Select the role from the drop-down list, and select **Add**.

Update the role entry:

1. Select **Manage > Role**, and select the role that you want to grant.
2. Select the **Role Members** tab, and select **Add Role Members**.
3. Select the user from the drop-down list, and select **Add**.

Grant roles dynamically

Grant a role *dynamically* by using one of the following methods:

- Use a condition, expressed as a query filter, in the role definition. If the condition is `true` for a particular member, that member is granted the role. Conditions can be used in both managed and internal roles.
- Use a custom script to define a more complex role-granting strategy.

Grant a role based on a condition

A role that is granted based on a defined condition is called a *conditional role*. To create a conditional role, include a query filter in the role definition.

> **Important**
>
> Properties that are used as the basis of a conditional role query *must* be configured as `searchable` and must be indexed in the repository configuration. To configure a property as `searchable`, update the property definition in your managed object configuration. For more information, refer to Create and modify object types.

To create a conditional role by using the admin UI, select **Condition** on the role **Details** page, then define the query filter that will be used to assess the condition.

To create a conditional role over REST, include the query filter as a value of the `condition` property in the role definition. The following example creates a role, `fr-employee`, that will be granted only to those users who live in France (whose `country` property is set to `FR`):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "name": "fr-employee",
  "description": "Role granted to employees resident in France",
  "condition": "/country eq \"FR\""
}' \
"http://localhost:8080/openidm/managed/role?_action=create"
{
  "_id": "eb18a2e2-ee1e-4cca-83fb-5708a41db94f",
  "_rev": "000000004085704c",
  "name": "fr-employee",
  "description": "Role granted to employees resident in France",
  "condition": "/country eq \"FR\""
}
```

When a conditional role is created or updated, IDM automatically assesses all managed users, and recalculates the value of their `roles` property, if they qualify for that role. When a condition is removed from a role, that is, when the role becomes an unconditional role, all conditional grants are removed. So, users who were granted the role based on the condition, have that role removed from their `roles` property.

Caution

When a conditional role is defined in an existing data set, every user entry (including the mapped entries on remote systems) must be updated with the assignments implied by that conditional role. The time that it takes to create a new conditional role is impacted by the following items:

- The number of managed users affected by the condition.
- The number of assignments related to the conditional role.
- The average time required to provision updates to all remote systems affected by those assignments.

In a data set with a very large number of users, creating a new conditional role can therefore incur a significant performance cost when you create it. Ideally, you should set up your conditional roles at the beginning of your deployment to avoid performance issues later.

Query a user's roles

To query user roles over REST, query the user's `roles` property. The following example shows that scarter has been granted two roles—an `employee` role, and an `fr-employee` role:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/scarter/roles?_queryFilter=true&_fields=_ref/*,name"
{
  "result": [
    {
      "_id": "5a023862-654d-4d7f-b9d0-7c151b8dede5",
      "_rev": "00000000baa999c1",
      "_refResourceCollection": "managed/role",
      "_refResourceId": "b8783543-869a-4bd4-907e-9c1d89f826ae",
      "_refResourceRev": "0000000027a959cf",
      "name": "employee",
      "_ref": "managed/role/b8783543-869a-4bd4-907e-9c1d89f826ae",
      "_refProperties": {
        "_id": "5a023862-654d-4d7f-b9d0-7c151b8dede5",
        "_rev": "00000000baa999c1"
      }
    },
    {
      "_id": "b281ffdf-477e-4211-a112-84476435bab2",
      "_rev": "00000000d612a248",
      "_refResourceCollection": "managed/role",
      "_refResourceId": "01ee6191-75d8-4d4b-9291-13a46592c57a",
      "_refResourceRev": "000000000cb0794d",
      "name": "fr-employee",
      "_ref": "managed/role/01ee6191-75d8-4d4b-9291-13a46592c57a",
      "_refProperties": {
        "_grantType": "conditional",
        "_id": "b281ffdf-477e-4211-a112-84476435bab2",
        "_rev": "00000000d612a248"
      }
    }
  ],
  ...
}
```

Note that the `fr-employee` role indicates a `_grantType` of `conditional`. This property indicates *how* the role was granted to the user. If no `_grantType` is listed, the role was granted statically.

Querying a user's roles in this way *does not* return any roles that would be in effect as a result of a custom script, or of any temporal constraint applied to the role. To return a complete list of *all* the roles in effect at a specific time, query the user's `effectiveRoles` property, as follows:

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

Alternatively, to check which roles have been granted to a user, either statically or dynamically, look at the user's entry in the admin UI:

1. Select **Manage > User**, then select the user whose roles you want to see.
2. Select the **Provisioning Roles** tab.
3. If you have a large number of managed roles, use the **Advanced Filter** option on the **Role List** page to build a custom query.

Delete a user's roles

To remove a statically granted role from a user entry, do one of the following:

- Update the value of the user's `roles` property to remove the reference to the role.
- Update the value of the role's `members` property to remove the reference to that user.

> **Important**
>
> A delegated administrator must use PATCH to add or remove relationships.
> Roles that have been granted as the result of a condition can only be removed when the condition is changed or removed, or when the role itself is deleted.

Over REST

Use one of the following methods to remove a role grant from a user:

- DELETE the role from the user's `roles` property, including the reference ID (the ID of the relationship between the user and the role) in the delete request.

  The following example removes the `employee` role from user scarter. The role ID is `b8783543-869a-4bd4-907e-9c1d89f826ae`, but the ID required in the DELETE request is the reference ID (`5a023862-654d-4d7f-b9d0-7c151b8dede5`):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/managed/user/scarter/roles/5a023862-654d-4d7f-b9d0-7c151b8dede5"
{
  "_id": "5a023862-654d-4d7f-b9d0-7c151b8dede5",
  "_rev": "00000000baa999c1",
  "_ref": "managed/role/b8783543-869a-4bd4-907e-9c1d89f826ae",
  "_refResourceCollection": "managed/role",
  "_refResourceId": "b8783543-869a-4bd4-907e-9c1d89f826ae",
  "_refProperties": {
    "_id": "5a023862-654d-4d7f-b9d0-7c151b8dede5",
    "_rev": "00000000baa999c1"
  }
}
```

- PATCH the user entry to remove the role from the array of roles, specifying the *value* of the role object in the JSON payload.

> **Caution**
>
> When you remove a role in this way, you must include the *entire object* in the value, as shown in the following example:

```
curl \
--header "Content-type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
  {
    "operation" : "remove",
    "field" : "/roles",
    "value" : {
      "_ref": "managed/role/b8783543-869a-4bd4-907e-9c1d89f826ae",
      "_refResourceCollection": "managed/role",
      "_refResourceId": "b8783543-869a-4bd4-907e-9c1d89f826ae",
      "_refProperties": {
        "_id": "5a023862-654d-4d7f-b9d0-7c151b8dede5",
        "_rev": "00000000baa999c1"
      }
    }
  }
]' \
"http://localhost:8080/openidm/managed/user/scarter"
{
  "_id": "scarter",
  "_rev": "000000007b78257d",
  "mail": "scarter@example.com",
  "givenName": "Steven",
  "sn": "Carter",
  "description": "Created By CSV",
  "userName": "scarter",
  "telephoneNumber": "1234567",
  "accountStatus": "active",
  "effectiveRoles": [
    {
      "_ref": "managed/role/01ee6191-75d8-4d4b-9291-13a46592c57a"
    }
  ],
  "effectiveAssignments": [],
  "preferences": {
    "updates": false,
    "marketing": false
  },
  "country": "France"
}
```

- DELETE the user from the role's `members` property, including the reference ID (the ID of the relationship between the user and the role) in the delete request.

  The following example first queries the members of the `employee` role, to obtain the ID of the relationship, then removes bjensen's membership from that role:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/role/b8783543-869a-4bd4-907e-9c1d89f826ae/members?
_queryFilter=true"
{
  "result": [
    {
      "_id": "a5a4bf94-6425-4458-aae4-bbd6ad094f72",
      "_rev": "00000000c25d994a",
      "_ref": "managed/user/bjensen",
      "_refResourceCollection": "managed/user",
      "_refResourceId": "bjensen",
      "_refProperties": {
        "_id": "a5a4bf94-6425-4458-aae4-bbd6ad094f72",
        "_rev": "00000000c25d994a"
      }
    }
  ],
  ...
}
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/managed/role/b8783543-869a-4bd4-907e-9c1d89f826ae/members/
a5a4bf94-6425-4458-aae4-bbd6ad094f72"
{
  "_id": "a5a4bf94-6425-4458-aae4-bbd6ad094f72",
  "_rev": "00000000c25d994a",
  "_ref": "managed/user/bjensen",
  "_refResourceCollection": "managed/user",
  "_refResourceId": "bjensen",
  "_refProperties": {
    "_id": "a5a4bf94-6425-4458-aae4-bbd6ad094f72",
    "_rev": "00000000c25d994a"
  }
}
```

Use the admin UI

Use one of the following methods to remove a user's roles:

Method 1:

1. Select **Manage > User**, and select the user whose roles you want to remove.
2. Select the **Provisioning Roles** tab, select the role that you want to remove, then select **Remove Selected Provisioning Roles**.

Method 2:

1. Select **Manage > Role**, and select the role whose members you want to remove.
2. Select the **Role Members** tab, select the members that you want to remove, then select **Remove Selected Role Members**.

Delete a role definition

To delete a role over the REST interface, simply delete that managed object. The following command deletes the `employee` role created in the previous section:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/managed/role/b8783543-869a-4bd4-907e-9c1d89f826ae"
{
  "_id": "b8783543-869a-4bd4-907e-9c1d89f826ae",
  "_rev": "0000000027a959cf",
  "privileges": [],
  "name": "employee",
  "description": "All employees"
}
```

Note

You cannot delete a role that is currently granted to users. If you attempt to delete a role that is granted to a user (either over the REST interface, or by using the admin UI), IDM returns an error. The following example attempts to remove a role that is still granted to a user:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/managed/role/01ee6191-75d8-4d4b-9291-13a46592c57a"
{
  "code": 409,
  "reason": "Conflict",
  "message": "Cannot delete a role that is currently granted"
}
```

To delete a role through the admin UI, select **Manage > Role**, select the role you want to remove, then click **Delete Selected**.

Use temporal constraints to restrict effective roles

Temporal constraints restrict the period that a role is effective. You can apply temporal constraints to managed and internal roles, and to role *grants* (for individual users).

For example, you might want a role, `contractors-2020`, to apply to all contract employees for the year 2020. In this case, you would set the temporal constraint on the role. Alternatively, you might want to assign a `contractors` role that applies to an individual user only for the period of their contract of employment.

The following examples show how to set temporal constraints on role definitions, and on individual role grants.

Add a temporal constraint to a role

When you create a role, you can include a temporal constraint in the role definition that restricts the validity of the role, regardless of how that role is granted. Temporal constraints are expressed as a time interval in ISO 8601 date and time format. For more information on this format, refer to the ISO 8601 standard.

The following example adds a `contractor` role over the REST interface. The role is effective from March 1st, 2020 to August 31st, 2020:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "name": "contractor",
  "description": "Role granted to contract workers for 2020",
  "temporalConstraints": [
    {
      "duration": "2020-03-01T00:00:00.000Z/2020-08-31T00:00:00.000Z"
    }
  ]
}' \
"http://localhost:8080/openidm/managed/role?_action=create"
{
  "_id": "ed761370-b24f-4e21-8e58-a3230942da67",
  "_rev": "000000007429750e",
  "name": "contractor",
  "description": "Role granted to contract workers for 2020",
  "temporalConstraints": [
    {
      "duration": "2020-03-01T00:00:00.000Z/2020-08-31T00:00:00.000Z"
    }
  ]
}
```

This example specifies the time zone as Coordinated Universal Time (UTC) by appending `Z` to the time. If no time zone information is provided, the time zone is assumed to be local time. To specify a different time zone, include an offset (from UTC) in the format `±hh:mm`. For example, an interval of `2020-03-01T00:00:00.000-07:00/2020-08-31T00:00:00.000-07:00` specifies a time zone that is seven hours behind UTC.

When the period defined by the constraint has ended, the role object remains in the repository, but the effective roles script will not include the role in the list of effective roles for any user.

The following example assumes that user scarter has been granted a role `contractor-march`. A temporal constraint has been included in the `contractor-march` role definition, specifying that the role should be applicable only during the month of March 2020. At the end of this period, a query on scarter's entry shows that his `roles` property still includes the `contractor-march` role (with ID `0face495-772d-4d36-a30d-8594618aba0d`), but his `effectiveRoles` property does not:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/scarter?_fields=_id,userName,roles,effectiveRoles"
{
  "_id": "scarter",
  "_rev": "00000000e5fdeb51",
  "userName": "scarter",
  "effectiveRoles": [],
  "roles": [
    {
      "_ref": "managed/role/0face495-772d-4d36-a30d-8594618aba0d",
      "_refResourceCollection": "managed/role",
      "_refResourceId": "0face495-772d-4d36-a30d-8594618aba0d",
      "_refProperties": {
        "_id": "5f41d5a5-19b4-4524-a4b1-445790ff14da",
        "_rev": "00000000cb339810"
      }
    }
  ]
}
```

The role is still in place but is no longer effective.

To restrict the period during which a role is valid by using the admin UI, select **Temporal Constraint** on the role **Details** tab, then select a timezone offset relative to GMT and the start and end dates for the required period.

Add a temporal constraint to a role grant

To restrict the validity of a role for individual users, apply a temporal constraint at the grant level, rather than as part of the role definition. In this case, the temporal constraint is taken into account per user, when the user's effective roles are calculated. Temporal constraints that are defined at the grant level can be different for each user who is a member of that role.

To apply a temporal constraint to a grant over the REST interface, include the constraint as one of the `_refProperties` of the relationship between the user and the role. The following example assumes a `contractor` role, with ID `ed761370-b24f-4e21-8e58-a3230942da67`. The command adds user bjensen as a member of that role, with a temporal constraint that specifies that she be a member of the role for one year only, from January 1st, 2020 to January 1st, 2021:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/members/-",
    "value": {
      "_ref": "managed/user/bjensen",
      "_refProperties": {
        "temporalConstraints": [{"duration": "2020-01-01T00:00:00.000Z/2021-01-01T00:00:00.000Z"}]
      }
    }
  }
]' \
"http://localhost:8080/openidm/managed/role/ed761370-b24f-4e21-8e58-a3230942da67"
{
  "_id": "ed761370-b24f-4e21-8e58-a3230942da67",
  "_rev": "000000007429750e",
  "name": "contractor",
  "description": "Role granted to contract workers for 2020",
  "temporalConstraints": [
    {
      "duration": "2020-03-01T00:00:00.000Z/2020-08-31T00:00:00.000Z"
    }
  ]
}
```

A query on bjensen's roles property shows that the temporal constraint has been applied to this grant:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/bjensen/roles?_queryFilter=true"
{
  "result": [
    {
      "_id": "40600260-111d-4695-81f1-450365025784",
      "_rev": "00000000173daedb",
      "_ref": "managed/role/ed761370-b24f-4e21-8e58-a3230942da67",
      "_refResourceCollection": "managed/role",
      "_refResourceId": "ed761370-b24f-4e21-8e58-a3230942da67",
      "_refProperties": {
        "temporalConstraints": [
          {
            "duration": "2020-01-01T00:00:00.000Z/2021-01-01T00:00:00.000Z"
          }
        ],
        "_id": "40600260-111d-4695-81f1-450365025784",
        "_rev": "00000000173daedb"
      }
    }
  ],
  ...
}
```

To restrict the period during which a role grant is valid using the admin UI, set a temporal constraint when you add the member to the role.

For example, to specify that bjensen be added to a **Contractor** role only for the period of her employment contract, select **Manage > Role**, select the **Contractor** role, then select **Add Role Members**. On the **Add Role Members** screen, select **bjensen** from the list, then enable the **Temporal Constraint**, and specify the start and end date of her contract.

Use assignments to provision users

Authorization roles control access to IDM itself. *Provisioning roles* define rules for how attribute values are updated on external systems. These rules are configured through *assignments* that are attached to a provisioning role definition. The purpose of an assignment is to provision an attribute or set of attributes, based on an object's role membership.

The synchronization mapping configuration between two resources provides the basic account provisioning logic (how an account is mapped from a source to a target system). Role assignments provide additional provisioning logic that is not covered in the basic mapping configuration. The attributes and values that are updated by using assignments might include group membership, access to specific external resources, and so on. A group of assignments can collectively represent a *role*.

Assignment objects are created, updated, and deleted like any other managed object, and are attached to a role by using the relationships mechanism, in much the same way as a role is granted to a user. Assignments are stored in the repository and are accessible at the context path `/openidm/managed/assignment`.

This section describes how to manipulate assignments over the REST interface, and by using the admin UI. When you have created an assignment, and attached it to a role definition, all user objects that reference that role definition will, as a result, reference the corresponding assignment in their `effectiveAssignments` attribute.

Important

If you have mapped roles and assignments to properties on a target system, and you are preloading the result set into memory, make sure that your `targetQuery` returns the mapped property. For example, if you have mapped a specific role to the `ldapGroups` property on the target system, the target query must include the `ldapGroups` property when it returns the object.
The following mapping excerpt indicates that the target query must return the `_id` of the object as well as its `ldapGroups` property:

```
"targetQuery": {
    "_queryFilter": true,
    "_fields": "_id,ldapGroups"
}
```

For more information about preloading the result set for reconciliation operations, refer to Improve Reconciliation Query Performance.

Create an Assignment

You can create assignments over the REST interface or using the admin UI:

Over REST

To create a new assignment over REST, send a PUT or POST request to the `/openidm/managed/assignment` context path.

The following example creates a new managed assignment named `employee`. The JSON payload in this example shows the following:

- The assignment is applied for the mapping `managedUser_systemLdapAccounts`, so attributes will be updated on the external LDAP system specified in this mapping.
- The name of the attribute on the external system whose value will be set is `employeeType`, and its value will be set to `Employee`.
- When the assignment is applied during a sync operation, the attribute value `Employee` is added to any existing values for that attribute. When the assignment is removed (if the role is deleted, or if the user is no longer a member of that role), the attribute value `Employee` is removed from the values of that attribute.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "name": "employee",
  "description": "Assignment for employees.",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "employeeType",
      "value": [
        "Employee"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ]
}' \
"http://localhost:8080/openidm/managed/assignment?_action=create"
{
  "_id": "1a6a3af3-024f-4cf1-b4f6-116b98053816",
  "_rev": "00000000b2329649",
  "name": "employee",
  "description": "Assignment for employees.",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "employeeType",
      "value": [
        "Employee"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ]
}
```

Note that at this stage, the assignment is not linked to any role, so no user can make use of the assignment. You must add the assignment to a role, as described in Add an Assignment to a Role.

Using the admin UI

1. Select **Manage > Assignment > New Assignment**.
2. Enter a name and description for the new assignment.
3. Select the mapping to which the assignment should apply. The mapping indicates the target resource, that is, the resource on which the attributes specified in the assignment will be adjusted.
4. Select **Save** to add the assignment.

5. Select the **Attributes** tab and select the attribute or attributes whose values will be adjusted by this assignment. The attribute you select here will determine what is displayed next:
    - Regular text field—specify what the value of the attribute should be, when this assignment is applied.
    - Item button—specify a managed object type, such as an object, relationship, or string.
    - Properties button—specify additional information, such as an array of role references.
6. Select the assignment operation from the dropdown list:
    - `Merge With Target`: the attribute value will be added to any existing values for that attribute. This operation merges the existing value of the target object attribute with the value(s) from the assignment. If duplicate values are found (for attributes that take a list as a value), each value is included only once in the resulting target. This assignment operation is used only with complex attribute values like arrays and objects, and does not work with strings or numbers.
    - `Replace Target`: the attribute value will overwrite any existing values for that attribute. The value from the assignment becomes the authoritative source for the attribute.
7. Select the unassignment operation from the drop-down list:
    - `Remove From Target`: the attribute value is removed from the system object when the user is no longer a member of the role, or when the assignment itself is removed from the role definition.
    - `No Operation`: removing the assignment from the user's `effectiveAssignments` has no effect on the current state of the attribute in the system object.
8. Select the **Events** tab to specify any scriptable events associated with this assignment.

    The assignment and unassignment operations described in the previous step operate at the *attribute level*. That is, you specify what should happen with each attribute affected by the assignment when the assignment is applied to a user, or removed from a user.

    The scriptable *On assignment* and *On unassignment* events operate at the *assignment level*, rather than the attribute level. Define scripts here to apply additional logic or operations that should be performed when a user (or other object) receives or loses an entire assignment. This logic can be anything that is not restricted to an operation on a single attribute.

    For information about the variables available to these scripts, refer to Variables available to role assignment scripts.
9. Select the **Roles** tab to attach this assignment to an existing role definition.

Attribute encryption on assignments

Assignment attributes are encrypted if the corresponding connector attribute indicates confidentiality, based on the attribute's `nativeType` (such as `JAVA_TYPE_GUARDEDSTRING` or `JAVA_TYPE_GUARDED_BYTE_ARRAY`).

The managed assignment object includes the following property:

```
"attributeEncryption" : { }
```

If `attributeEncryption` is not present on managed/assignment, the assignment attributes are not encrypted. If the property is present but empty, it will default to IDM's default encryption cipher. To specify a different cipher, add the `cipher` property. For example:

```
"attributeEncryption" : {
  "cipher" : "AES/CBC/PKCS5Padding"
}
```

This functionality makes use of the `idm.assignment.attribute.encryption` secret, found in `secrets.json`.

Add an assignment to a role

After you have created a role, and an assignment, you create a *relationship* between the assignment and the role, in much the same way as a user references a role.

Update a role definition to include one or more assignments over the REST interface, or by using the admin UI:

Over REST

Update the role definition to include a reference to the ID of the assignment in the `assignments` property of the role. The following example adds the `employee` assignment (ID `1a6a3af3-024f-4cf1-b4f6-116b98053816`) to an existing `employee` role (ID `2243f5f8-ed75-4c3b-b4b3-058d5c58fbb4`):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/assignments/-",
    "value": { "_ref": "managed/assignment/1a6a3af3-024f-4cf1-b4f6-116b98053816" }
  }
]' \
"http://localhost:8080/openidm/managed/role/2243f5f8-ed75-4c3b-b4b3-058d5c58fbb4"
{
  "_id": "2243f5f8-ed75-4c3b-b4b3-058d5c58fbb4",
  "_rev": "00000000e85263c7",
  "privileges": [],
  "name": "employee",
  "description": "Roll granted to all permanent employees"
}
```

To check that the assignment was added successfully, query the role's `assignments` property:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/role/2243f5f8-ed75-4c3b-b4b3-058d5c58fbb4/assignments?
_queryFilter=true&_fields=_ref/*,name,assignments"
{
  "result": [
    {
      "_id": "d15822f0-05bc-464a-927d-8e5018a234d3",
      "_rev": "0000000010eea343",
      "_refResourceCollection": "managed/assignment",
      "_refResourceId": "1a6a3af3-024f-4cf1-b4f6-116b98053816",
      "_refResourceRev": "00000000b2329649",
      "name": "employee",
      "_ref": "managed/assignment/1a6a3af3-024f-4cf1-b4f6-116b98053816",
      "_refProperties": {
        "_id": "d15822f0-05bc-464a-927d-8e5018a234d3",
        "_rev": "0000000010eea343"
      }
    }
  ],
  ...
}
```

Note that the `assignments` property references the assignment that you created in the previous step.

To remove an assignment from a role definition, remove the reference to the assignment from the role's `assignments` property.

Using the admin UI

1. Select **Manage > Role**, and select the role to which you want to add an assignment.
2. Select the **Managed Assignments** tab, and select **Add Managed Assignments**.
3. Select the assignment that you want to add to the role, then select **Add**.

Delete an assignment

Delete assignments over the REST interface, or by using the admin UI:

Over REST

To delete an assignment over the REST interface, simply delete that object. The following example deletes the `employee` assignment created in the previous example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/managed/assignment/1a6a3af3-024f-4cf1-b4f6-116b98053816"
{
  "_id": "1a6a3af3-024f-4cf1-b4f6-116b98053816",
  "_rev": "00000000b2329649",
  "name": "employee",
  "description": "Assignment for employees.",
  "mapping": "managedUser_systemLdapAccounts",
  "attributes": [
    {
      "name": "employeeType",
      "value": [
        "Employee"
      ],
      "assignmentOperation": "mergeWithTarget",
      "unassignmentOperation": "removeFromTarget"
    }
  ]
}
```

Note

You *can* delete an assignment, even if it is referenced by a managed role. When the assignment is removed, any users to whom the corresponding roles were granted will no longer have that assignment in their list of `effectiveAssignments`. For more information about effective roles and effective assignments, refer to Effective roles and effective assignments.

Using the admin UI

To delete an assignment by using the admin UI, select **Manage > Assignment**.

Select the assignment you want to remove, then select **Delete**.

Effective roles and effective assignments

Effective roles and *effective assignments* are virtual properties of a user object. Their values are calculated by IDM, using relationships between related objects to know when to recalculate when changes occur. The relationships between objects are configured using the `notify`, `notifySelf`, and `notifyRelationships` settings for `managed/user`, `managed/role`, and `managed/assignment`. Which related objects to traverse for calculation is configured using `queryConfig`. Calculation or recalculation is triggered when the roles or assignments for a managed user are added, removed, or changed, including by changes from temporal constraints, and notification of that change is sent to the related objects.

The following excerpt of the managed object configuration file shows how these two virtual properties are constructed for each managed user object:

```
"effectiveRoles" : {
    "type" : "array",
    "title" : "Effective Roles",
    "description" : "Effective Roles",
    "viewable" : false,
    "returnByDefault" : true,
    "isVirtual" : true,
    "queryConfig" : {
        "referencedRelationshipFields" : ["roles"]
    },
    "usageDescription" : "",
    "isPersonal" : false,
    "items" : {
        "type" : "object",
        "title" : "Effective Roles Items"
    }
},
"effectiveAssignments" : {
    "type" : "array",
    "title" : "Effective Assignments",
    "description" : "Effective Assignments",
    "viewable" : false,
    "returnByDefault" : true,
    "isVirtual" : true,
    "queryConfig" : {
        "referencedRelationshipFields" : ["roles", "assignments"],
        "referencedObjectFields" : ["*"]
    },
    "usageDescription" : "",
    "isPersonal" : false,
    "items" : {
        "type" : "object",
        "title" : "Effective Assignments Items"
    }
}
```

When a role references an assignment, and a user references the role, that user automatically references the assignment in its list of effective assignments.

`effectiveRoles` uses the `roles` relationship to calculate the grants that are currently in effect, including any qualified by temporal constraints.

`effectiveAssignments` uses the `roles` relationship, and the `assignments` relationship for each role, to calculate the current assignments in effect for that user. The synchronization engine reads the calculated value of the `effectiveAssignments` attribute when it processes the user. The target system is updated according to the configured `assignmentOperation` for each assignment.

When a user's roles or assignments are updated, IDM calculates the `effectiveRoles` and `effectiveAssignments` for that user based on the current value of the user's `roles` property, and the `assignments` property of any roles referenced by the `roles` property. The previous set of examples showed the creation of a role `employee` that referenced an assignment `employee` and was granted to user bjensen. Querying that user entry would show the following effective roles and effective assignments:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/bjensen?
_fields=userName,roles,effectiveRoles,effectiveAssignments"
{
  "_id": "ca8855fd-a404-42c7-88b7-02f8a8a825b2",
  "_rev": "0000000081eebe1a",
  "userName": "bjensen",
  "effectiveRoles": [
    {
      "_refResourceCollection": "managed/role",
      "_refResourceId": "2243f5f8-ed75-4c3b-b4b3-058d5c58fbb4"
      "_ref": "managed/role/2243f5f8-ed75-4c3b-b4b3-058d5c58fbb4"
    }
  ],
  "effectiveAssignments": [
    {
      "name": "employee",
      "description": "Assignment for employees.",
      "mapping": "managedUser_systemLdapAccounts",
      "attributes": [
        {
          "assignmentOperation": "mergeWithTarget",
          "name": "employeeType",
          "unassignmentOperation": "removeFromTarget",
          "value": [
            "employee"
          ]
        }
      ],
      "_rev": "0000000087d5a9a5",
      "_id": "46befacf-a7ad-4633-864d-d93abfa561e9"
      "_refResourceCollection": "managed/assignment",
      "_refResourceId": "46befacf-a7ad-4633-864d-d93abfa561e9",
      "_ref": "managed/assignment/46befacf-a7ad-4633-864d-d93abfa561e9"
    }
  ],
  "roles": [
    {
      "_ref": "managed/role/2243f5f8-ed75-4c3b-b4b3-058d5c58fbb4",
      "_refResourceCollection": "managed/role",
      "_refResourceId": "2243f5f8-ed75-4c3b-b4b3-058d5c58fbb4",
      "_refProperties": {
        "_id": "93552530-10fa-49a4-865f-c942dffd2801",
        "_rev": "0000000081ed9f2b"
      }
    }
  ]
}
```

In this example, synchronizing the managed/user repository with the external LDAP system defined in the mapping populates user bjensen's `employeeType` attribute in LDAP with the value `employee`.

Roles and relationship change notification

Before you read this section, refer to Configure relationship change notification to understand the `notify` and `notifyRelationships` properties, and how change notification works for relationships in general. In the case of roles, the change notification configuration exists to ensure that managed users are notified when any of the relationships that link users, roles, and assignments are manipulated (that is, created, updated, or deleted).

Consider the situation where a user has role `R`. A new assignment `A` is created that references role `R`. Ultimately, we want to notify all users that have role `R` so that their reconciliation state will reflect any attributes in the new assignment `A`. We achieve this notification with the following configuration:

In the managed object schema, the `assignment` object definition has a `roles` property that includes a `resourceCollection`. The `path` of this resource collection is `managed/role` and `"notify" : true` for the resource collection:

```
{
    "name" : "assignment",
    "schema" : {
        ...
        "properties" : {
            ...
            "roles" : {
                ...
                "items" : {
                    ...
                    "resourceCollection" : [
                        {
                            "notify" : true,
                            "path" : "managed/role",
                            "label" : "Role",
                            "query" : {
                                "queryFilter" : "true",
                                "fields" : [
                                    "name"
                                ]
                            }
                        }
                    ...
}
```

With this configuration, when assignment `A` is created, with a reference to role `R`, role `R` is notified of the change. However, we still need to propagate that notification to any `users` who are `members` of role `R`. To do this, we configure the `role` object as follows:

```
{
    "name" : "role",
    "schema" : {
        ...
        "properties" : {
            ...
            "assignments" : {
                ...
                "notifyRelationships" : ["members"]
            }
    ...
}
```

When role `R` is notified of the creation of a new relationship to assignment `A`, the notification is propagated through the `assignments` property. Because `"notifyRelationships" : ["members"]` is set on the `assignments` property, the notification is propagated across role `R` to all members of role `R`.

Managed role script hooks

Like any other managed object, you can use script hooks to configure role behavior.

Map roles to external groups

A user's access to IDM is based on one or more *authorization roles*. Authorization roles are cumulative, and are calculated for a user in the following order:

1. Roles set specifically in the user's `userRoles` property
2. Group roles — based on group membership in an external system

   Group roles are controlled with the following properties in the authentication configuration:

   - `groupMembership`: the property on the external system that represents group membership. In a DS directory server, that property is `ldapGroups` by default. In an Active Directory server, the property is `memberOf` by default. For example:

     ```
     "groupMembership" : "ldapGroups"
     ```

     Note that the value of the `groupMembership` property must be the OpenICF property name defined in the provisioner file, rather than the property name on the external system.

   - `groupRoleMapping`: a mapping between an IDM role and a group on the external system. Setting this property ensures that if a user authenticates through pass-through authentication, they are given specific IDM roles depending on their membership in groups on the external system. In the following example, users who are members of the group `cn=admins,ou=Groups,dc=example,dc=com` are given the internal `openidm-admin` role when they authenticate:

```
"groupRoleMapping" : {
    "internal/role/openidm-admin" : ["cn=admins,ou=Groups,dc=example,dc=com"]
}
```

- `groupComparisonMethod`: the method used to check whether the authenticated user's group membership matches one of the groups mapped to an IDM role (in the `groupRoleMapping` property).

  The `groupComparisonMethod` can be one of the following:

  - `equals`: a case-sensitive equality check
  - `caseInsensitive`: a case-insensitive equality check
  - `ldap`: a case-insensitive and whitespace-insensitive equality check. Because LDAP directories do not take case or whitespace into account in group DNs, you must set the `groupComparisonMethod` if you are using pass-through authentication with an LDAP directory.

> **Note**
>
> To control access to *external systems*, use *provisioning roles* and assignments, as described in Use assignments to provision users.

Organizations

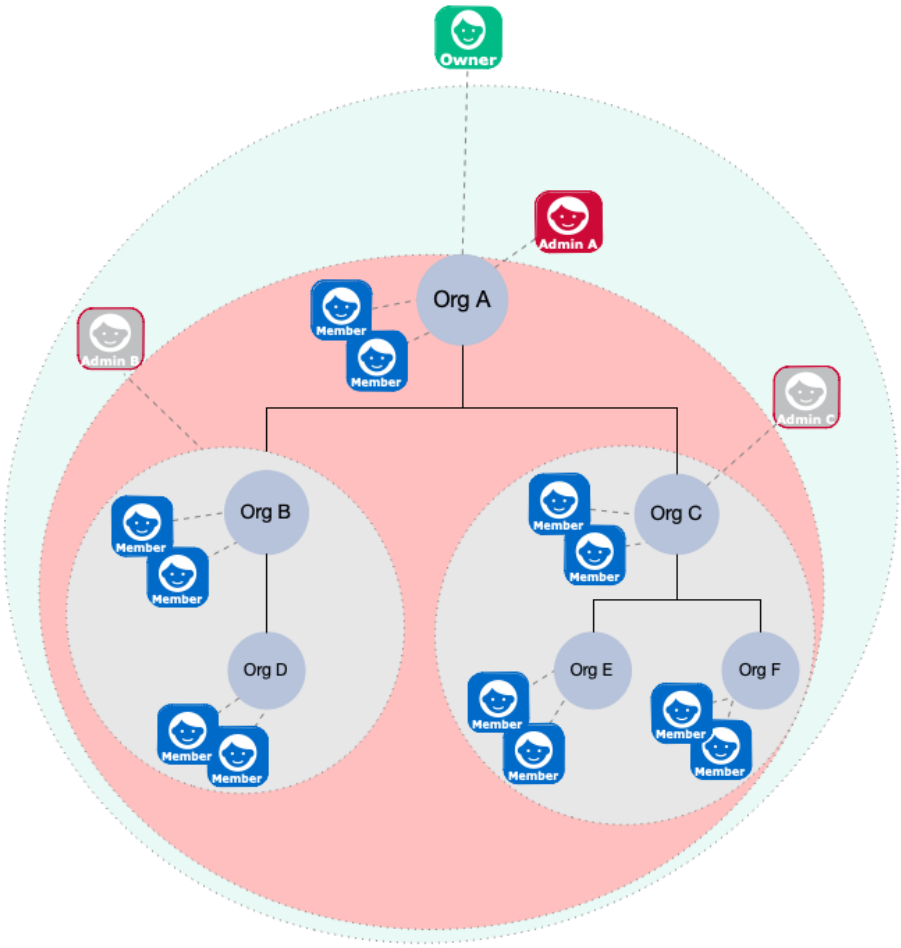
Organization objects let you arrange and manage users in *hierarchical trees*. Organizations also let you give users fine-grained administrative privileges to various parts of the tree based on their location in that tree. For example, an administrator of one organization might have full access to the users within that organization but no access to the users in an adjacent organization.

An organization object (defined in the managed object configuration) contains a set of relationship properties that reference the parent, child, owners, admins, and members of an organization. These relationship properties enable a hierarchical organizational model.

Users and organizations have a set of relationship-derived virtual properties used by the delegated administration filters to provide the visibility and access constraints that underpin the organization model. Users have the `ids` of all the organizations of which they are members, and organizations have the `ids` of all their admin and owner users.

Only IDM administrative users can create top-level organizations. Within organizations, there are various levels of privileges, depending on how a user is *related* to the organization.

The following diagram gives a high-level overview of how privileges are assigned to various entities in the organization hierarchy:



- An *organization owner* can manipulate all organizations, members, and admins in their *ownership area*. The ownership area includes any part of the tree in or beneath the organization that they own. So, in the preceding image, the owner of **Org A** can do the following anywhere within their ownership area (the pale green region):
  - Add and update members.

    > **Note**
    >
    > Organization owners only have access to the members in their ownership area. So, an owner can *create* a new user as a member of their organization, but cannot add an existing managed user to their organization if that user is outside of their ownership area (that is, in any part of the tree not in or beneath the organization that they own).

  - Add and update sub-organizations, such as **Org B** and **Org C**.
  - Give an organization member the `admin` privilege for the parent organization or any sub-organizations.

    An organization owner *cannot* create additional *owners* in their root organization.

> **Caution**
>
> An organization owner does not have to be a *member* of the organization. If the organization owner *is* a member of the organization, that owner is automatically in the administrative area of any admins of that organization, and can therefore be manipulated by an organization admin. To avoid accidentally giving organization admins privileges over an organization owner, do not make the owner a member of the organization.

- An *organization admin* has control over their *administrative area*. The administrative area includes any part of the tree in or beneath the organization that they administer. In the preceding diagram, the administrative area of Admin A is shaded red. The administrative areas of Admins B and C are shaded grey. An admin can do the following in their administrative area:
  - Add and update members.

    > **Note**
    >
    > Organization admins only have access to the members in their administrative area. So, an admin can *create* a new user as a member of their organization, but cannot add an existing managed user to their organization if that user is outside of their administrative area (that is, in any part of the tree not in or beneath the organization that they administer).

  - Add and update sub-organizations of the organization they administer.

    Notice that Admin B and C are outside of the administrative area of Admin A. An organization admin *cannot* create additional *admins* in their administrative areas.

    An organization admin *must* be a member of the organization, so must either be an existing member of the organization, or must be given the `memberOfOrg` relationship at the time they are created.
- *Organization members* are regular users, with no special privileges in the organization hierarchy.

  Managed users have a `memberOfOrgIDs` virtual property that lists the organizations to which the user belongs (either directly, or through any parent organizations).
- *Parent and child organizations* are essentially relationships between an organization and existing organizations in the tree.

Manage organizations over REST

IDM provides RESTful access to managed organizations, at the context path `/openidm/managed/organization`. You can add, change, and delete organizations by using the admin UI or over the REST interface. To use the admin UI, select **Manage > Organization**.

The following examples show how to add, change, and delete organizations over the REST interface. For a reference of all managed organization endpoints and actions, refer to Managed organizations.

Only IDM administrators can create top level organizations.

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "name": "example-org"
}' \
"http://localhost:8080/openidm/managed/organization/example-org"
{
  "_id": "example-org",
  "_rev": "00000000bc9871c8",
  "adminIDs": [],
  "ownerIDs": [],
  "parentAdminIDs": [],
  "parentIDs": [],
  "parentOwnerIDs": [],
  "name": "example-org"
}
```

IDM administrators can create owners for an organization. This example makes bjensen the owner of the organization created previously. The example assumes that the managed user bjensen already exists:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{"_ref":"managed/user/bjensen"}' \
"http://localhost:8080/openidm/managed/organization/example-org/owners?_action=create"
{
  "_id": "fcb0f4d0-dad2-4138-a80c-62407a8e831e",
  "_rev": "00000000496d9920",
  "_ref": "managed/user/bjensen",
  "_refResourceCollection": "managed/user",
  "_refResourceId": "bjensen",
  "_refProperties": {
    "_id": "fcb0f4d0-dad2-4138-a80c-62407a8e831e",
    "_rev": "00000000496d9920"
  }
}
```

This example lists the organizations of which bjensen is an owner:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/bjensen/ownerOfOrg?_queryFilter=true"
{
  "result": [
    {
      "_id": "fcb0f4d0-dad2-4138-a80c-62407a8e831e",
      "_rev": "00000000496d9920",
      "_ref": "managed/organization/example-org",
      "_refResourceCollection": "managed/organization",
      "_refResourceId": "example-org",
      "_refProperties": {
        "_id": "fcb0f4d0-dad2-4138-a80c-62407a8e831e",
        "_rev": "00000000496d9920"
      }
    }
  ],
  ...
}
```

Organization owners can create members in the organizations that they own. In this example bjensen creates user scarter and makes him a member of the organization created previously:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Th3Password" \
--header "Accept-API-Version: resource=1.0" \
--request PUT \
--data '{
  "userName": "scarter",
  "sn": "Carter",
  "givenName": "Steven",
  "mail": "scarter@example.com",
  "password": "Th3Password",
  "memberOfOrg": [{"_ref": "managed/organization/example-org"}]
}' \
"http://localhost:8080/openidm/managed/user/scarter"
{
  "_id": "scarter",
  "_rev": "00000000eac81c23"
}
```

Organization owners can view the members of the organizations that they own. In this example, bjensen lists the members of example-org:

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Th3Password" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/organization/example-org/members?_queryFilter=true"
{
  "result": [
    {
      "_id": "b71e8dd9-6224-466f-9630-4358a69c69fd",
      "_rev": "0000000038ea999e",
      "_ref": "managed/user/scarter",
      "_refResourceCollection": "managed/user",
      "_refResourceId": "scarter",
      "_refProperties": {
        "_id": "b71e8dd9-6224-466f-9630-4358a69c69fd",
        "_rev": "0000000038ea999e"
      }
    }
  ],
  ...
}
```

Organization owners can create admins of the organizations that they own. An organization admin must be a member of the organization. In this example, bjensen makes scarter an admin of example-org:

```
curl \
--header 'Content-Type: application/json' \
--header "Accept-API-Version: resource=1.0" \
--header 'X-OpenIDM-Username: bjensen' \
--header 'X-OpenIDM-Password: Th3Password' \
--request PATCH \
--data '[
    {
        "operation": "add",
        "field": "/admins/-",
        "value": {
            "_ref": "managed/user/scarter"
        }
    }
]' \
"http://localhost:8080/openidm/managed/organization/example-org"
{
  "_id": "example-org",
  "_rev": "000000009c248a4a",
  "adminIDs": [
    "scarter"
  ],
  "ownerIDs": [
    "bjensen"
  ],
  "parentAdminIDs": [],
  "parentIDs": [],
  "parentOwnerIDs": [],
  "name": "example-org"
}
```

An organization owner or admin can only access the organizations that they own or administer. In this example, the admin scarter lists the organizations, and accesses only those of which they are an admin:

```
curl \
--header "X-OpenIDM-Username: scarter" \
--header "X-OpenIDM-Password: Th3Password" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/organization?_queryFilter=true"
{
  "result": [
    {
      "_id": "example-org",
      "_rev": "000000009c248a4a",
      "adminIDs": [
        "scarter"
      ],
      "ownerIDs": [
        "bjensen"
      ],
      "parentAdminIDs": [],
      "parentIDs": [],
      "parentOwnerIDs": [],
      "name": "example-org"
    }
  ],
  ...
}
```

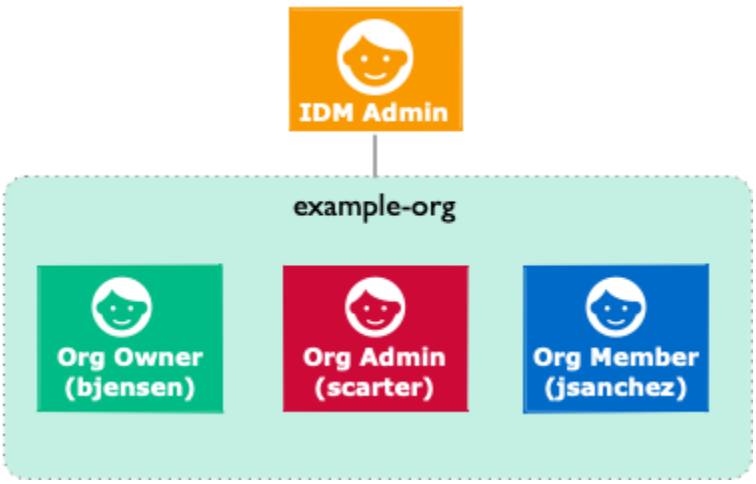
Organization admins can also add members to the organizations they administer. In this example, the organization admin, scarter, creates a new member, jsanchez, and makes her a member of example-org:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: scarter" \
--header "X-OpenIDM-Password: Th3Password" \
--header "Accept-API-Version: resource=1.0" \
--request PUT \
--data '{
  "userName": "jsanchez",
  "sn": "Sanchez",
  "givenName": "Juanita",
  "mail": "jsanchez@example.com",
  "password": "Th3Password",
  "memberOfOrg": [{"_ref": "managed/organization/example-org"}]
}' \
"http://localhost:8080/openidm/managed/user/jsanchez"
{
  "_id": "jsanchez",
  "_rev": "00000000f9341bd6"
}
```

Organization owners and admins can list the organizations of which a user is a member, as long as those organizations are owned or administrated by them. In this example, scarter lists the organizations of which jsanchez is a member:

```
curl \
--header "X-OpenIDM-Username: scarter" \
--header "X-OpenIDM-Password: Th3Password" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/jsanchez?_fields=memberOfOrg"
{
  "_id": "jsanchez",
  "_rev": "00000000f9341bd6",
  "memberOfOrg": [
    {
      "_ref": "managed/organization/example-org",
      "_refResourceCollection": "managed/organization",
      "_refResourceId": "example-org",
      "_refProperties": {
        "_id": "078d14b2-e5f1-4b21-9801-041138e691f4",
        "_rev": "00000000ac2e9927"
      }
    }
  ]
}
```

The organization established by the previous set of examples can be represented as follows:



In this organization, both bjensen and scarter can create and delete sub-organizations, also known as *child organizations*, of example-org, and can create and delete members within these child organizations.

The following example shows how to add and delete child organizations over the REST interface:

Organization owners and admins can create and manage child organizations of the organizations that they own or administer. In this example, the organization owner, bjensen, creates a new organization named `example-child-org`, and makes it a child organization of `example-org`:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Th3Password" \
--header "Accept-API-Version: resource=1.0" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "name": "example-child-org",
  "parent": {"_ref": "managed/organization/example-org"}
}' \
"http://localhost:8080/openidm/managed/organization/example-child-org"
{
  "_id": "example-child-org",
  "_rev": "00000000db852a9d"
}
```

Note

The organization model is based on delegated administration. As with delegated administration, you cannot explicitly change the relationship endpoints. So, for example, so you cannot create, update, delete, or patch relationship edges. The following type of request is therefore *not possible* with the organization model:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Th3Password" \
--header "Accept-API-Version: resource=1.0" \
--header "If-None-Match: *" \
--request PUT \
--data '{
"name": "example-child-org",
"parent": {"_ref": "managed/organization/example-org"}
}' \
"http://localhost:8080/openidm/managed/organization/children?_action=create"
```

Organization owners and admins have access to any organizations that are *child* organizations of their own orgs. In this example, admin scarter lists his visible organizations again:

```
curl \
--header "X-OpenIDM-Username: scarter" \
--header "X-OpenIDM-Password: Th3Password" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/organization?_queryFilter=true"
{
  "result": [
    {
      "_id": "example-org",
      "_rev": "000000009c248a4a",
      "adminIDs": [
        "scarter"
      ],
      "ownerIDs": [
        "bjensen"
      ],
      "parentAdminIDs": [],
      "parentIDs": [],
      "parentOwnerIDs": [],
      "name": "example-org"
    },
    {
      "_id": "example-child-org",
      "_rev": "00000000db852a9d",
      "adminIDs": [],
      "ownerIDs": [],
      "parentAdminIDs": [
        "scarter"
      ],
      "parentIDs": [
        "example-org"
      ],
      "parentOwnerIDs": [
        "bjensen"
      ],
      "name": "example-child-org"
    }
  ],
  ...
}
```

Notice that scarter can now access the example-child-org that bjensen created in the previous example.

Organizations in high latency environments

The relationship-derived virtual properties that support the organization model are generally calculated in response to relationship signals that travel *down* the organization tree hierarchy. Imagine, for example, that a new root organization is added to an existing organization hierarchy (or that a new admin or owner is added to the root of an existing organization hierarchy). The relationship signals that trigger relationship-derived virtual property calculation are propagated down the organization hierarchy, and to all members of the organizations in this hierarchy. This, in turn, updates their relationship-derived virtual property state.

If there are many thousands of members of the organizations in the hierarchy, this operation can take a long time to complete. It is therefore best practice to grow an organization hierarchy *downwards*, adding new organizations as leaves to an existing hierarchy, and adding new admins and members to the leaves in the hierarchy tree. This is preferable to growing the hierarchy *upwards*, starting with the leaves, and growing the hierarchy up towards the root.

If you *must* add a new root to an existing organization hierarchy with many organizations and many members, or a new admin or owner to an organization near the top of the hierarchy, rather perform this request over the command-line, using the examples provided in the previous section.

Use policies to validate data

IDM provides an extensible policy service that lets you apply specific validation requirements to various components and properties. This chapter describes the policy service, and provides instructions on configuring policies for managed objects.

The policy service provides a REST interface for reading policy requirements and validating the properties of components against configured policies. Objects and properties are validated automatically when they are created, updated, or patched. Policies are generally applied to user passwords, but can also be applied to any managed or system object, and to internal user objects.

The policy service lets you accomplish the following tasks:

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

The router service limits policy application to managed and internal user objects. To apply policies to additional objects, such as the audit service, modify your project's router configuration. For more information about the router service, refer to Script triggers defined in the router configuration.

A configurable default policy applies to all managed objects.

Default policy for managed objects

Policies applied to managed objects are configured in two places:

- A policy script that defines each policy and specifies how policy validation is performed.

  For more information, refer to Policy Script.

- A managed object policy element, defined in your managed object configuration, that specifies which policies are applicable to each managed resource. For more information, refer to Policy Configuration Element.

> **Note**
>
> The policy configuration determines which policies apply to resources *other than managed objects*. The default policy configuration includes policies that are applied to internal user objects, but you can extend the configuration to apply policies to system objects.

Policy script

The policy script file (`openidm/bin/defaults/script/policy.js`) separates policy configuration into two parts:

- A policy configuration object, which defines each element of the policy. For more information, refer to Policy Configuration Objects.
- A policy implementation function, which describes the requirements that are enforced by that policy.

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

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

policyFunctions.atLeastXCapitalLetters = function(fullObject, value, params, property) {
    var isRequired = _.find(this.failedPolicyRequirements, function (fpr) {
            return fpr.policyRequirement === "REQUIRED";
        }),
        isString = (typeof(value) === "string"),
        valuePassesRegexp = (function (v) {
            var test = isString ? v.match(/[A-Z]/g) : null;
            return test !== null && test.length >= params.numCaps;
        }(value));

    if ((isRequired || isString) && !valuePassesRegexp) {
        return [ { "policyRequirement" : "AT_LEAST_X_CAPITAL_LETTERS", "params" : {"numCaps": params.numCaps} } ];
    }

    return [];
}
...
```

To enforce user passwords that contain at least one capital letter, the `policyId` from the preceding example is applied to the appropriate resource (`managed/user/*`). The required number of capital letters is defined in the policy configuration element of the managed object configuration file (see Policy Configuration Element).

Policy configuration objects

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

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

`policyId`	A unique ID that enables the policy to be referenced by component objects.
`policyExec`	The name of the function that contains the policy implementation. For more information, refer to Policy Implementation Functions.
`clientValidation`	Indicates whether the policy decision can be made on the client. When `"clientValidation": true`, the source code for the policy decision function is returned when the client requests the requirements for a property.
`validateOnlyIfPresent`	Notes that the policy is to be validated only if the field within the object being validated exists.
`policyRequirements`	An array containing the policy requirement ID of each requirement that is associated with the policy. Typically, a policy will validate only one requirement, but it can validate more than one.

Policy implementation functions

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

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

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

The following example shows the implementation function for the `required` policy:

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

Default policy reference

IDM includes the following default policies and parameters:

Policy Id	Parameters	
`required` The property is required; not optional.		
`not-empty` The property can't be empty.		
`not-null` The property can't be null.		
`unique` The property must be unique.		
`valid-username` Tests for uniqueness and internal user conflicts.		
`no-internal-user-conflict` Tests for internal user conflicts.		
`regexpMatches` Matches a regular expression.	`regexp` `flags`	The regular expression pattern.
`valid-type` Tests for the specified types.	`types`	
`valid-query-filter` Tests for a valid query filter.		
`valid-array-items` Tests for valid array items.		
`valid-date` Tests for a valid date.		
`valid-formatted-date` Tests for a valid date format.		

Policy Id	Parameters	
`valid-time` Tests for a valid time.		
`valid-datetime` Tests for a valid date and time.		
`valid-duration` Tests for a valid duration format.		
`valid-email-address-format` Tests for a valid email address.		
`valid-name-format` Tests for a valid name format.		
`valid-phone-format` Tests for a valid phone number format.		
`at-least-X-capitals` The property must contain the minimum specified number of capital letters.	`numCaps`	Minimum number of capital letters.
`at-least-X-numbers` The property must contain the minimum specified number of numbers.	`numNums`	Minimum number of numbers.
`validNumber` Tests for a valid number.		
`minimumNumber` The property value must be greater than the `minimum`.	`minimum`	The minimum value.
`maximumNumber` The property value must be less than the `maximum`.	`maximum`	The maximum value.
`minimum-length` The property's minimum string length.	`minLength`	The minimum string length.
`maximum-length` The property's maximum string length.	`maxLength`	The maximum string length.
`cannot-contain-others` The property cannot contain values of the specified fields.	`disallowedFields`	A comma-separated list of the fields to check against. For example, the default managed user password policy specifies `userName,givenName,sn` as disallowed fields.

Policy Id	Parameters	
`cannot-contain-characters` The property cannot contain the specified characters.	`forbiddenChars`	A comma-separated list of disallowed characters. For example, the default managed user `userName` policy specifies `/` as a disallowed character.
`cannot-contain-duplicates` The property cannot contain duplicate characters.		
`mapping-exists` A sync mapping must exist for the property.		
`valid-permissions` Tests for valid permissions.		
`valid-accessFlags-object` Tests for valid access flags.		
`valid-privilege-path` Tests for a valid privilege path.		
`valid-temporal-constraints` Tests for valid temporal constraints.		

Policy configuration element

Properties defined in the managed object configuration can include a `policies` element that specifies how policy validation should be applied to that property. The following excerpt of the default managed object configuration shows how policy validation is applied to the `password` and `_id` properties of a managed/user object.

> **Note**
>
> You can only declare policies on top-level managed object attributes. Nested attributes (those within an array or object) cannot have policy declared on them.

```
{
    "name" : "user",
    "schema" : {
        "id" : "http://jsonschema.net",
        "properties" : {
            "_id" : {
                "description" : "User ID",
                "type" : "string",
                "viewable" : false,
                "searchable" : false,
                "userEditable" : false,
                "usageDescription" : "",
                "isPersonal" : false,
                "policies" : [
                    {
                        "policyId" : "cannot-contain-characters",
                        "params" : {
                            "forbiddenChars" : [
                                "/"
                            ]
                        }
                    }
                ]
            },
            "password" : {
                "title" : "Password",
                "description" : "Password",
                "type" : "string",
                "viewable" : false,
                "searchable" : false,
                "userEditable" : true,
                "encryption" : {
                    "purpose" : "idm.password.encryption"
                },
                "scope" : "private",
                "isProtected": true,
                "usageDescription" : "",
                "isPersonal" : false,
                "policies" : [
                    {
                        "policyId" : "minimum-length",
                        "params" : {
                            "minLength" : 8
                        }
                    },
                    {
                        "policyId" : "at-least-X-capitals",
                        "params" : {
                            "numCaps" : 1
                        }
                    },
                    {
                        "policyId" : "at-least-X-numbers",
                        "params" : {
                            "numNums" : 1
                        }
                    },
                    {
                        "policyId" : "cannot-contain-others",
                        "params" : {
```

```
                            "disallowedFields" : [
                                "userName",
                                "givenName",
                                "sn"
                            ]
                        }
                    }
                ]
            }
        }
    }
}
```

Note that the policy for the `_id` property references the function `cannot-contain-characters`, that is defined in the `policy.js` file. The policy for the `password` property references the functions `minimum-length`, `at-least-X-capitals`, `at-least-X-numbers`, and `cannot-contain-others`, that are defined in the `policy.js` file. The parameters that are passed to these functions (number of capitals required, and so forth) are specified in the same element.

Validate managed object data types

The `type` property of a managed object specifies the data type of that property, for example, `array`, `boolean`, `number`, `null`, `object`, or `string`. For more information about data types, refer to the JSON Schema Primitive Types section of the JSON Schema standard.

The `type` property is subject to policy validation when a managed object is created or updated. Validation fails if data does not match the specified `type`, such as when the data is an `array` instead of a `string`. The default `valid-type` policy enforces the match between property values and the `type` defined in the managed object configuration.

IDM supports multiple valid property types. For example, you might have a scenario where a managed user can have more than one telephone number, or a *null* telephone number (when the user entry is first created and the telephone number is not yet known). In such a case, you could specify the accepted property type as follows in your managed object configuration:

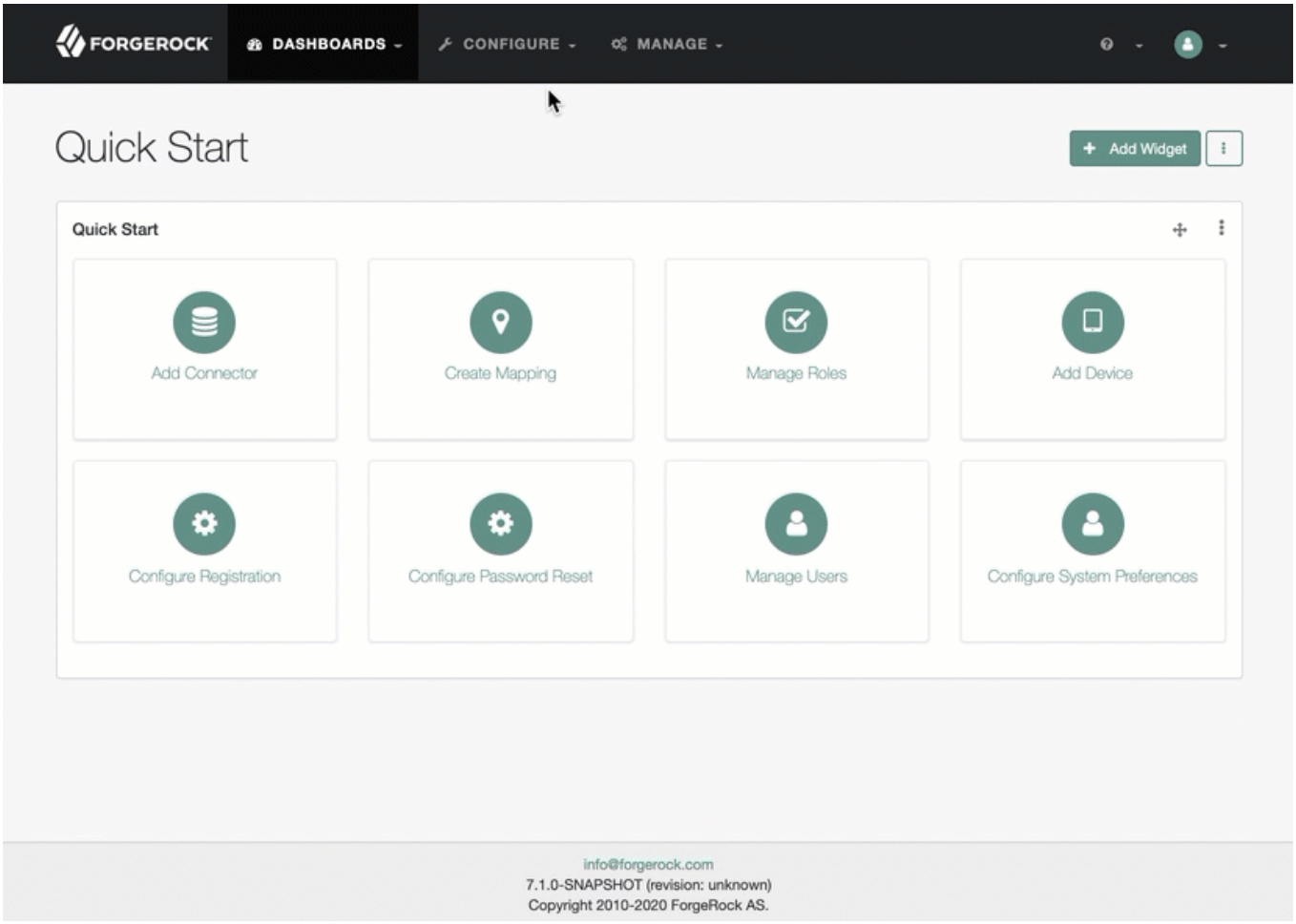
```
"telephoneNumber" : {
    "type" : "string",
    "title" : "Telephone Number",
    "description" : "Telephone Number",
    "viewable" : true,
    "userEditable" : true,
    "pattern" : "^\\+?([0-9\\- \\(\\)])*$",
    "usageDescription" : "",
    "isPersonal" : true,
    "policies" : [
        {
            "policyId" : "minimum-length",
            "params" : {
                "minLength" : 1
            }
        },
        {
            "policyId": "maximum-length",
            "params": {
                "maxLength": 255
            }
        }
    ]
}
```

In this case, the `valid-type` policy from the `policy.js` file checks the telephone number for an accepted `type` and `pattern`, either for a real telephone number or a `null` entry.

Configure policy validation using the admin UI

To configure policy validation for a managed object type using the admin UI, update the configuration of the object type—a high-level overview:

1. Go to the managed object, and edit or create a property.
2. Click the **Validation** tab, and add the policy.



1. From the navigation bar, click **Configure > Managed Objects**.
2. On the **Managed Objects** page, edit or create a managed object.
3. On the **Managed Object NAME** page, do one of the following:
    - To edit an existing property, click the property.
    - To create a property, click **Add a Property**, enter the required information, and click **Save**.
        - Now click the property.
4. From the **Validation** tab, click **Add Policy**.
5. In the **Add/Edit Policy** window, enter information in the following fields, and click **Add** or **Save**:

Policy Id

Refers to the unique `PolicyId` in the `policy.js` file.

For a list of the default policies, refer to Policy Reference.

Parameter Name

Refers to the parameters for the `PolicyId`. For a list of the default policy parameters, refer to Policy Reference.

Value

The parameter's value to validate.

Important

Be cautious when using Validation Policies. If a policy relates to an array of relationships, such as between a user and multiple devices, **Return by Default** should always be set to `false`. You can verify this in your managed object configuration. Any managed object that has `items` of `"type" : "relationship"`, must also have `"returnByDefault" : false`.

Extend the policy service

You can extend the policy service by adding custom scripted policies, and by adding policies that are applied only under certain conditions.

Add custom scripted policies

If your deployment requires additional validation functionality that is not supplied by the default policies, you can add your own policy scripts to your project's `script` directory, and reference them in your project's policy configuration.

Do not modify the default policy script file (`openidm/bin/defaults/script/policy.js`) as doing so might result in interoperability issues in a future release.

To reference additional policy scripts, set the `additionalFiles` property in you policy configuration.

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

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

addPolicy(policy);

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

The `mypolicy.js` policy is referenced in the `policy.json` configuration file as follows:

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

> **Note**
>
> In cases where you are using the admin UI, both `policy.js` and `mypolicy.js` will be run within the client, and then again by the the server. When creating new policies, be aware that these policies may be run in both contexts.

Add conditional policy definitions

You can extend the policy service to support policies that are applied only under specific conditions. To apply a conditional policy to managed objects, add the policy to your project's managed object configuration. To apply a conditional policy to other objects, add it to your project's policy configuration.

The following managed object configuration shows a sample conditional policy for the `password` property of managed user objects. The policy indicates that sys-admin users have a more lenient password policy than regular employees:

```
{
    "objects" : [
        {
            "name" : "user",
            ...
                "properties" : {
                ...
                    "password" : {
                        "title" : "Password",
                        "type" : "string",
                        ...
                        "conditionalPolicies" : [
                            {
                                "condition" : {
                                    "type" : "text/javascript",
                                    "source" : "(fullObject.org === 'sys-admin')"
                                },
                                "dependencies" : [ "org" ],
                                "policies" : [
                                    {
                                        "policyId" : "max-age",
                                        "params" : {
                                            "maxDays" : ["90"]
                                        }
                                    }
                                ]
                            },
                            {
                                "condition" : {
                                    "type" : "text/javascript",
                                    "source" : "(fullObject.org === 'employees')"
                                },
                                "dependencies" : [ "org" ],
                                "policies" : [
                                    {
                                        "policyId" : "max-age",
                                        "params" : {
                                            "maxDays" : ["30"]
                                        }
                                    }
                                ]
                            }
                        ],
                        "fallbackPolicies" : [
                            {
                                "policyId" : "max-age",
                                "params" : {
                                    "maxDays" : ["7"]
                                }
                            }
                        ]
                    }
                    ...
}
```

To understand how a conditional policy is defined, examine the components of this sample policy. For more information on the policy function, refer to Policy Implementation Functions.

There are two distinct scripted conditions (defined in the `condition` elements). The first condition asserts that the user object, contained in the `fullObject` argument, is a member of the `sys-admin` org. If that assertion is true, the `max-age` policy is applied to the `password` attribute of the user object, and the maximum number of days that a password may remain unchanged is set to `90`.

The second condition asserts that the user object is a member of the `employees` org. If that assertion is true, the `max-age` policy is applied to the `password` attribute of the user object, and the maximum number of days that a password may remain unchanged is set to `30`.

In the event that neither condition is met (the user object is not a member of the `sys-admin` org or the `employees` org), an optional fallback policy can be applied. In this example, the fallback policy also references the `max-age` policy and specifies that for such users, their password must be changed after 7 days.

The `dependencies` field prevents the condition scripts from being run at all, if the user object does not include an `org` attribute.

Note

This example assumes that a custom `max-age` policy validation function has been defined, as described in Add Custom Scripted Policies.

Tip

These scripted conditions do not apply to progressive profiling.

Disable policy enforcement

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

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

You should not disable policy enforcement permanently, in a production environment.

Manage policies over REST

Manage the policy service over the REST interface at the `openidm/policy` endpoint.

List the defined policies

The following REST call displays a list of all the policies defined in `policy.json` (policies for objects other than managed objects). The policy objects are returned in JSON format, with one object for each defined policy ID:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/policy"
{
  "_id": "",
  "resources": [
    ...
    {
      "resource": "internal/user/*",
      "properties": [
        {
          "name": "_id",
          "policies": [
            {
              "policyId": "cannot-contain-characters",
              "params": {
                "forbiddenChars": [ "/" ]
              },
              "policyFunction": "\nfunction (fullObject, value, params, property) {\n    ...",
              "policyRequirements": [
                "CANNOT_CONTAIN_CHARACTERS"
              ]
            }
          ],
          "policyRequirements": [
            "CANNOT_CONTAIN_CHARACTERS"
          ]
        }
        ...
      ]
      ...
    }
  ]
}
```

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/policy/managed/user/*"
{
  "_id": "*",
  "resource": "managed/user/*",
  "properties": [
    {
      "policyRequirements": [
        "VALID_TYPE",
        "CANNOT_CONTAIN_CHARACTERS"
      ],
      "fallbackPolicies": null,
      "name": "_id",
      "policies": [
        {
          "policyRequirements": [
            "VALID_TYPE"
          ],
          "policyId": "valid-type",
          "params": {
            "types": [
              "string"
            ]
          }
        },
        {
          "policyId": "cannot-contain-characters",
          "params": {
            "forbiddenChars": [ "/" ]
          },
          "policyFunction": "...",
          "policyRequirements": [
            "CANNOT_CONTAIN_CHARACTERS"
          ]
        }
      ],
      "conditionalPolicies": null
    }
    ...
  ]
}
```

Validate objects and properties over REST

To verify that an object adheres to the requirements of all applied policies, include the `validateObject` action in the request.

The following example verifies that a new managed user object is acceptable, in terms of the policy requirements. Note that the ID in the URL (`test` in this example) is ignored—the action simply validates the object in the JSON payload:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "sn": "Jones",
  "givenName": "Bob",
  "telephoneNumber": "0827878921",
  "passPhrase": null,
  "mail": "bjones@example.com",
  "accountStatus": "active",
  "userName": "bjones@example.com",
  "password": "123"
}' \
"http://localhost:8080/openidm/policy/managed/user/test?_action=validateObject"
{
  "result": false,
  "failedPolicyRequirements": [
    {
      "policyRequirements": [
        {
          "policyRequirement": "MIN_LENGTH",
          "params": {
            "minLength": 8
          }
        }
      ],
      "property": "password"
    },
    {
      "policyRequirements": [
        {
          "policyRequirement": "AT_LEAST_X_CAPITAL_LETTERS",
          "params": {
            "numCaps": 1
          }
        }
      ],
      "property": "password"
    }
  ]
}
```

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

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

The following example checks whether a user's new password (`12345`) is acceptable:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "password": "12345"
}' \
"http://localhost:8080/openidm/policy/managed/user/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb?
_action=validateProperty"
{
  "result": false,
  "failedPolicyRequirements": [
    {
      "policyRequirements": [
        {
          "policyRequirement": "MIN_LENGTH",
          "params": {
            "minLength": 8
          }
        }
      ],
      "property": "password"
    },
    {
      "policyRequirements": [
        {
          "policyRequirement": "AT_LEAST_X_CAPITAL_LETTERS",
          "params": {
            "numCaps": 1
          }
        }
      ],
      "property": "password"
    }
  ]
}
```

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

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "password": "1NewPassword"
}' \
"http://localhost:8080/openidm/policy/managed/user/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb?
_action=validateProperty"
{
  "result": true,
  "failedPolicyRequirements": []
}
```

Validate field removal

To validate field removal, specify the fields to remove when calling the policy `validateProperty` action. You cannot remove fields that:

- Are required in the `required` schema array.
- Have a `required` policy.
- Have a default value.

The following example validates the removal of the fields `description` and `givenName`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "_remove": [ "description", "givenName" ]
}' \
"http://localhost:8080/openidm/policy/managed/user/ca5a3196-2ed3-4a76-8881-30403dee70e9?
_action=validateProperty"
```

Because `givenName` is a required field, IDM returns a failed policy validation:

```
{
  "result": false,
  "failedPolicyRequirements": [
    {
      "policyRequirements": [
        {
          "policyRequirement": "REQUIRED"
        }
      ],
      "property": "givenName"
    }
  ]
}
```

Force validation of default values

IDM does not perform policy validation for default values specified in the managed objects schema. It may be necessary to force validation when validating properties for an object that does not yet exist. To force validation, include `forceValidate=true` in the request URL.

Validate properties to unknown resource paths

To perform a `validateProperty` action to a path that is unknown (`*`), such as `managed/user/*` or `managed/user/userDoesntExistYet` , the payload must include:

- An `object` field that contains the object details.
- A `properties` field that contains the properties to be evaluated.

Pre-registration validation example

A common use case for validating properties for unknown resources is prior to object creation, such as during pre-registration.

1. Always pass the object and properties content in the POST body because IDM has no object to look up.
2. Use any placeholder id in the request URL, as `*` has no special meaning in the API.

   This example uses a conditional policy for any object with the description `test1` :

```
"password" : {
    ...
    "conditionalPolicies" : [
        {
            "condition" : {
                "type" : "text/javascript",
                "source" : "(fullObject.description === 'test1')"
            },
            "dependencies" : [ "description" ],
            "policies" : [
                {
                    "policyId" : "at-least-X-capitals",
                    "params" : {
                        "numCaps" : 1
                    }
                }
            ]
        }
    ],
```

3. Using the above conditional policy, you could perform a `validateProperty` action to `managed/user/*` with the request:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "object": {
    "description": "test1"
  },
  "properties": {
    "password": "passw0rd"
  }
}' \
"http://localhost:8080/openidm/policy/managed/user/*?_action=validateProperty"
{
  "result": false,
  "failedPolicyRequirements": [
    {
      "policyRequirements": [
        {
          "params": {
            "numCaps": 1
          },
          "policyRequirement": "AT_LEAST_X_CAPITAL_LETTERS"
        }
      ],
      "property": "password"
    }
  ]
}
```

Store managed objects in the repository

IDM stores managed objects, internal users, and configuration objects in a repository. By default, the server uses an embedded ForgeRock Directory Services (DS) instance as its repository.

In production, you must replace this embedded instance with an external DS instance, or with a JDBC repository, as described in Select a repository.

These topics describe the repository configuration, and how objects are mapped in the repository.

Repository configuration files

Configuration files for all supported repositories are located in the `/path/to/openidm/db/database/conf` directory. For JDBC repositories, the configuration is defined in two files:

- `datasource.jdbc-default.json` specifies the connection to the database.
- `repo.jdbc.json` specifies the mapping between IDM resources and database tables.

For a DS repository, the `repo.ds.json` file specifies the resource mapping and, in the case of an external repository, the connection details to the LDAP server.

For both DS and JDBC, the `conf/repo.init.json` file specifies IDM's initial internal roles and users.

Copy the configuration files for your specific database type to your project's `conf/` directory.

JDBC connection configuration

The default database connection configuration file for a MySQL database follows:

```
{
    "driverClass" : "com.mysql.cj.jdbc.Driver",
    "jdbcUrl" : "jdbc:mysql://&{openidm.repo.host}:&{openidm.repo.port}/openidm?
allowMultiQueries=true&characterEncoding=utf8&serverTimezone=UTC",
    "databaseName" : "openidm",
    "username" : "openidm",
    "password" : "openidm",
    "connectionTimeout" : 30000,
    "connectionPool" : {
        "type" : "hikari",
        "minimumIdle" : 20,
        "maximumPoolSize" : 50
    }
}
```

The configuration file includes the following properties:

`driverClass`

`"driverClass" : string`

To use the JDBC driver manager to acquire a data source, set this property, as well as `jdbcUrl`, `username`, and `password`. The driver class must be the fully-qualified class name of the database driver to use for your database.

Using the JDBC driver manager to acquire a data source is the most likely option, and the only one supported "out of the box". The remaining options in the sample repository configuration file assume that you are using a JDBC driver manager.

Example: `"driverClass" : "com.mysql.cj.jdbc.Driver"`

jdbcUrl

The connection URL to the JDBC database. The URL should include all parameters required by your database. For example, to specify the encoding in MySQL use `'characterEncoding=utf8'`.

Specify the values for `openidm.repo.host` and `openidm.repo.port` in one of the following ways:

- Set the values in `resolver/boot.properties` or your project's `conf/system.properties` file, for example:

```
openidm.repo.host = localhost
openidm.repo.port = 3306
```

- Set the properties in the `OPENIDM_OPTS` environment variable and export that variable before startup. You must include the JVM memory options when you set this variable. For example:

```
export OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -
Dopenidm.repo.port=3306"
/path/to/openidm/startup.sh
Executing ./startup.sh...
Using OPENIDM_HOME:   /path/to/openidm
Using PROJECT_HOME:   /path/to/openidm
Using OPENIDM_OPTS:   -Xmx1024m -Xms1024m -Dopenidm.repo.host=localhost -
Dopenidm.repo.port=3306
Using LOGGING_CONFIG: -Djava.util.logging.config.file=/path/to/openidm/conf/logging.properties
Using boot properties at /path/to/openidm/resolver/boot.properties
-> OpenIDM version "8.0.0"
OpenIDM ready
```

databaseName

The name of the database, used in SQL queries. For example:

```
select * from databaseName.managedobjects
```

In addition to the SQL queries that are generated by IDM, any queries defined in the `repo.jdbc.json` file replace `{_dbSchema}` with the value of the `databaseName` property. For example, the following query in the `repo.jdbc.json` file replaces the `{_dbSchema}` with the value of the `databaseName`:

```
"delete-mapping-links" : "DELETE FROM ${_dbSchema}.${_table} WHERE linktype = ${mapping}",
```

`username`

The username with which to access the JDBC database.

`password`

The password with which to access the JDBC database. IDM automatically encrypts clear string passwords. To replace an existing encrypted value, replace the whole `crypto-object` value, including the brackets, with a string of the new password.

`connectionTimeout`

The period of time, in milliseconds, after which IDM should consider an attempted connection to the database to have failed. The default period is 30000 milliseconds (30 seconds).

`connectionPool`

Database connection pooling configuration. The default connection pool library is HikariCP:

```
"connectionPool" : {
    "type" : "hikari"
}
```

IDM uses the default HikariCP configuration, except for the following parameters. You might need to adjust these parameters, according to your database workload:

- `minimumIdle`

  This property controls the minimum number of idle connections that HikariCP maintains in the connection pool. If the number of idle connections drops below this value, HikariCP attempts to add additional connections.

  By default, HikariCP runs as a fixed-sized connection pool, that is, this property is not set. The connection configuration files provided with IDM set the minimum number of idle connections to `20`.

- `maximumPoolSize`

  This property controls the maximum number of connections to the database, including idle connections and connections that are being used.

  By default, HikariCP sets the maximum number of connections to `10`. The connection configuration files provided with IDM set the maximum number of connections to `50`.

For information about the HikariCP configuration parameters, refer to the HikariCP Project Page.

JDBC database table configuration

An excerpt of a MySQL database table configuration file follows:

```
{
    "dbType" : "MYSQL",
    "useDataSource" : "default",
    "maxBatchSize" : 100,
    "maxTxRetry" : 5,
    "queries" : {...},
    "commands" : {...},
    "resourceMapping" : {...}
}
```

The configuration file includes the following properties:

`dbType` *: string, optional*

The type of database. The database type might affect the queries used and other optimizations. Supported database types include the following:
`DB2`
`SQLSERVER` (for Microsoft SQL Server)
`MYSQL`
`ORACLE`
`POSTGRESQL`

`useDataSource` *: string, optional*

This option refers to the connection details that are defined in the configuration file, described previously. The default configuration file is named `datasource.jdbc-default.json`. This is the file that is used by default (and the value of the `"useDataSource"` is therefore `"default"`). You might want to specify a different connection configuration file, instead of overwriting the details in the default file. In this case, set your connection configuration file `datasource.jdbc-name.json` and set the value of `"useDataSource"` to whatever name you have used.

`maxBatchSize`

The maximum number of SQL statements that will be batched together. This parameter lets you optimize the time taken to execute multiple queries. Certain databases do not support batching, or limit how many statements can be batched. A value of `1` disables batching.

`maxTxRetry`

The maximum number of times that a specific transaction should be attempted before that transaction is aborted.

`queries`

Any custom queries that can be referenced from the configuration.

Options supported for query parameters include the following:

- A default string parameter, for example:

```
openidm.query("managed/user", { "_queryId": "for-userName", "uid": "jdoe" });
```

For more information about the query function, refer to openidm.query.

- A list parameter (`${list:propName}`).

  Use this parameter to specify a set of indeterminate size as part of your query. For example:

  ```
  WHERE targetObjectId IN (${list:filteredIds})
  ```

- A boolean parameter (`${bool:propName}`).

  Use this parameter to query boolean values in the database.

- Numeric parameters for integers (`${int:propName}`), large integers (`${long:propName}`), and decimal values (`${num:propName}`).

  Use these parameters to query numeric values in the database, corresponding to the column data type in your repository.

`commands`

Specific commands configured to manage the database over the REST interface. Currently, the following default commands are included in the configuration:

- `purge-by-recon-expired`
- `purge-by-recon-number-of`
- `delete-mapping-links`
- `delete-target-ids-for-recon`

These commands assist with removing stale reconciliation audit information from the repository, and preventing the repository from growing too large. The commands work by executing a query filter, then performing the specified operation on each result set. Currently the only supported operation is `DELETE` , which removes all entries that match the filter.

`resourceMapping`

Defines the mapping between IDM resource URIs (for example, `managed/user`) and JDBC tables. The structure of the resource mapping is as follows:

```
"resourceMapping" : {
    "default" : {
        "mainTable" : "genericobjects",
        "propertiesTable" : "genericobjectproperties",
        "searchableDefault" : true
    },
    "genericMapping" : {...},
    "explicitMapping" : {...}
}
```

The default mapping object represents a default generic table in which any resource that does not have a more specific mapping is stored.

The generic and explicit mapping objects are described in the following section.

DS repository configuration

An excerpt of a DS repository configuration file follows:

```
{
    "embedded" : false,
    "maxConnectionAttempts" : 5,
    "security" : {...},
    "ldapConnectionFactories" : {...},
    "queries" : {...},
    "commands" : {...},
    "rest2LdapOptions" : {...},
    "indices" : {...},
    "schemaProviders" : {...},
    "resourceMapping" : {...}
}
```

The configuration file includes the following properties:

`embedded` *: boolean*

Specifies an embedded or external DS instance.

IDM uses an embedded DS instance by default. The embedded instance is not supported in production.

`maxConnectionAttempts` *: integer*

Specifies the number of times IDM should attempt to connect to the DS instance. On startup, IDM will attempt to connect to DS indefinitely. The `maxConnectionAttempts` parameter controls the number of reconnection attempts in the event of a failure during normal operation, for example, if an attempt to access the DS repository times out.

By default, IDM will attempt to reconnect to the DS instance `5` times.

`security`

Specifies the keystore and truststore for secure connections to DS.

```
"security": {
    "trustManager": "file",
    "fileBasedTrustManagerType": "JKS",
    "fileBasedTrustManagerFile": "&{idm.install.dir}/security/truststore",
    "fileBasedTrustManagerPasswordFile": "&{idm.install.dir}/security/storepass"
}
```

In the default case, where DS servers use TLS key pairs generated using a deploymentId and deploymentIdPassword, you must import the deploymentId-based CA certificate into the IDM truststore. For more information, refer to External DS repository.

Note that the `security` settings have no effect for an embedded DS repository. Embedded DS is not supported in production, and is meant for evaluation or testing purposes only.

`ldapConnectionFactories`

For an external DS repository, configures the connection to the DS instance. For example:

```
"ldapConnectionFactories": {
  "bind": {
    "connectionSecurity": "startTLS",
    "heartBeatIntervalSeconds": 60,
    "heartBeatTimeoutMilliSeconds": 10000,
    "primaryLdapServers": [
      {
        "hostname": "localhost",
        "port": 31389
      }
    ],
    "secondaryLdapServers": []
  },
  "root": {
    "inheritFrom": "bind",
    "authentication": {
      "simple": { "bindDn": "uid=admin", "bindPassword": "password" }
    }
  }
}
```

The connection to the DS repository uses the DS *REST2LDAP* gateway and the `ldapConnectionFactories` property sets the gateway configuration. For example, the `secondaryLdapServers` property specifies an array of LDAP servers that the gateway can contact if the primary LDAP servers cannot be contacted.

For information on all the gateway configuration properties, refer to Gateway Configuration in the *DS REST API Guide*.

`queries`

Predefined queries that can be referenced from the configuration. For a DS repository, all predefined queries are really filtered queries (using the `_queryFilter` parameter), for example:

```
"query-all-ids": {
    "_queryFilter": "true",
    "_fields": "_id,_rev"
}
```

The queries are divided between those for `generic` mappings and those for `explicit` mappings, but the queries themselves are the same for both mapping types.

`commands`

Specific commands configured to manage the repository over the REST interface. Currently, only two commands are included by default:

- `delete-mapping-links`
- `delete-target-ids-for-recon`

Both of these commands assist with removing stale reconciliation audit information from the repository, and preventing the repository from growing too large.

`rest2LdapOptions`

Specifies the configuration for accessing the LDAP data stored in DS. For more information, refer to Gateway REST2LDAP Configuration in the *DS REST API Guide*.

`indices`

For generic mappings, enables you to set up LDAP indices on custom object properties. For more information, refer to Improving Generic Mapping Search Performance (DS).

`schemaProviders`

For generic mappings, enables you to list custom objects whose properties should be indexed. For more information, refer to Improving Generic Mapping Search Performance (DS).

`resourceMapping`

Defines the mapping between IDM resource URIs (for example, `managed/user`) and the DS directory tree. The structure of the resource mapping object is as follows:

```
{
    "resourceMapping" : {
        "defaultMapping": {
            "dnTemplate": "ou=generic,dc=openidm,dc=forgerock,dc=com"
        },
        "explicitMapping" : {...},
        "genericMapping" : {...}
    }
}
```

The default mapping object represents a default generic organizational unit (`ou`) in which any resource that does not have a more specific mapping is stored.

The generic and explicit mapping objects are described in Object mappings.

Object mappings

You can map IDM objects to the tables in a JDBC database or to organizational units in DS using:

Generic Mapping

Lets you store arbitrary objects without special configuration or administration.

Explicit Mapping

Maps specific objects and properties to tables and columns in the JDBC database or to organizational units in DS.

Hybrid Mapping

Similar to *Generic Mapping*, except some objects and properties are mapped explicitly.

By default, IDM uses a generic mapping for user-definable objects, for both a JDBC and a DS repository. A generic mapping speeds up initial deployment, and can make system maintenance more flexible by providing a stable database structure. In a test environment, generic tables let you modify the user and object model easily, without database access, and without the need to constantly add and drop table columns. However, generic mapping does not take full advantage of the underlying database facilities, such as validation within the database and flexible indexing. Using an explicit mapping generally results in a *substantial* performance improvement. It is therefore strongly advised that you change to an explicit mapping before deploying in a production environment. If you are integrating IDM with AM, and using a shared DS repository, you *must* use an explicit schema mapping.

Mapping strategies are discussed in the following sections, with separate topics for JDBC DS repositories.

Mappings with a JDBC repository

Generic, explicit, and hybrid, oh my!

Reasons for choosing generic or hybrid over explicit mappings include:

- Generic and hybrid mapped objects offer the flexibility to add and subtract non-searchable properties without having to modify the Database Data Definition Language (DDL) or IDM object configuration.
- The properties table for generic objects can grow large quickly.

  Consider that a single object with 10 searchable properties would populate 10 rows within the generic properties table. Performance can be increased if commonly searched properties are mapped to a single column in the object table. In addition, the datatype of the property value can be enforced by the DDL of the column, or perhaps a required field could be marked as `NOT NULL`. However, once a property is mapped to an explicit column, future changes to the property mapping may require a DDL change and possibly, a migration effort.

> **Note**
>
> PostgreSQL offers JSON capabilities that automatically makes all properties searchable. Although indexes will likely still need to be created for properties that need a performance boost.

Generic mappings (JDBC)

Generic mapping speeds up development, and can make system maintenance more flexible by providing a stable database structure. However, generic mapping can have a performance impact and does not take full advantage of the database facilities (such as validation within the database and flexible indexing). In addition, queries can be more difficult to set up.

In a generic table, the entire object content is stored in a single large-character field named `fullobject` in the `mainTable` for the object. To search on specific fields, you can read them by referring to them in the corresponding `properties table` for that object. The disadvantage of generic objects is that, because every property you might like to filter by is stored in a separate table, you must join to that table each time you need to filter by anything.

The following diagram shows a pared down database structure for the default generic tables, when using a MySQL repository. The diagram indicates the relationship between the main table and the corresponding properties table for each object.

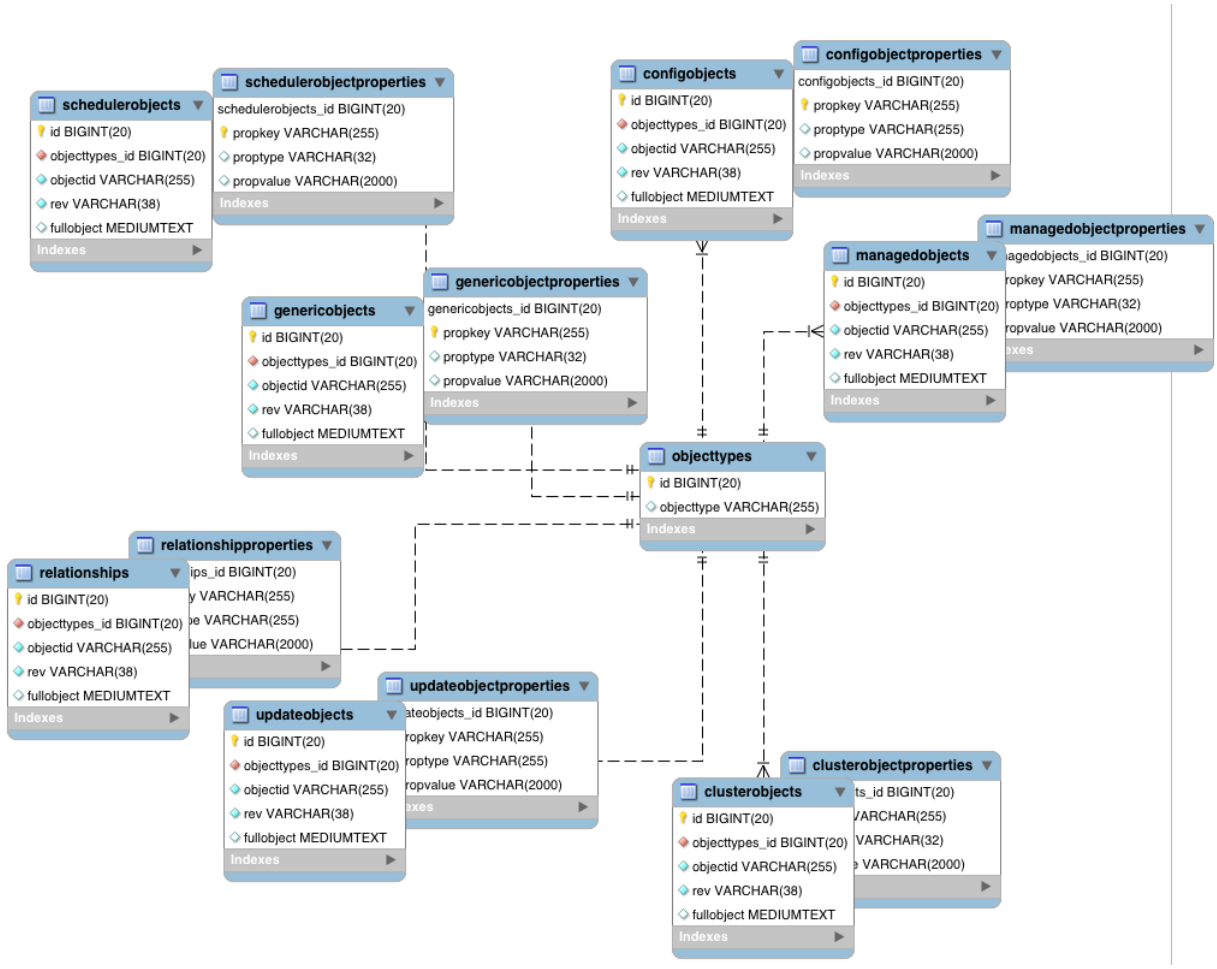


Figure 1. Generic Tables Entity Relationship Diagram

These separate tables can make the query syntax particularly complex. For example, a simple query to return user entries based on a user name would need to be implemented as follows:

```
SELECT obj.objectid, obj.rev, obj.fullobject FROM ${_dbSchema}.${_mainTable} obj
INNER JOIN ${_dbSchema}.${_propTable} prop ON obj.id = prop.${_mainTable}_id
INNER JOIN ${_dbSchema}.objecttypes objtype ON objtype.id = obj.objecttypes_id
WHERE prop.propkey='/userName' AND prop.propvalue = ${uid} AND objtype.objecttype = ${_resource}
```

The query can be broken down as follows:

1. Select the full object, the object ID, and the object revision from the main table:

```
SELECT obj.objectid, obj.rev, obj.fullobject FROM ${_dbSchema}.${_mainTable} obj
```

2. Join to the properties table and locate the object with the corresponding ID:

```
INNER JOIN ${_dbSchema}.${_propTable} prop  ON obj.id = prop.${_mainTable}_id
```

3. Join to the object types table to restrict returned entries to objects of a specific type. For example, you might want to restrict returned entries to `managed/user` objects, or `managed/role` objects:

```
INNER JOIN ${_dbSchema}.objecttypes objtype ON objtype.id = obj.objecttypes_id
```

4. Filter records by the `userName` property, where the userName is equal to the specified `uid` and the object type is the specified type (in this case, managed/user objects):

```
WHERE prop.propkey='/userName'
 AND prop.propvalue = ${uid}
 AND objtype.objecttype = ${_resource}
```

   The value of the `uid` field is provided as part of the query call, for example:

```
openidm.query("managed/user", { "_queryId": "for-userName", "uid": "jdoe" });
```

Tables for user definable objects use a generic mapping by default.

The following sample generic mapping object illustrates how `managed/` objects are stored in a generic table:

```
"genericMapping" : {
    "managed/*" : {
        "mainTable" : "managedobjects",
        "propertiesTable" : "managedobjectproperties",
        "searchableDefault" : true,
        "properties" : {
            "/picture" : {
                "searchable" : false
            }
        }
    }
}
```

`mainTable` *(string, mandatory)*

Indicates the main table in which data is stored for this resource.

The complete object is stored in the `fullobject` column of this table. The table includes an `objecttypes` foreign key that is used to distinguish the different objects stored within the table. In addition, the revision of each stored object is tracked, in the `rev` column of the table, enabling multiversion concurrency control (MVCC). For more information, refer to Manipulating Managed Objects Programmatically.

`propertiesTable` *(string, mandatory)*

Indicates the properties table, used for searches.

> **Note**
>
> PostgreSQL repositories do not use these properties tables to access specific properties. Instead, the PostgreSQL `json_extract_path_text()` function achieves this functionality.

The contents of the properties table is a defined subset of the properties, copied from the character large object (CLOB) that is stored in the `fullobject` column of the main table. The properties are stored in a one-to-many style separate table. The set of properties stored here is determined by the properties that are defined as `searchable`.

The stored set of searchable properties makes these values available as discrete rows that can be accessed with SQL queries, specifically, with `WHERE` clauses. It is not otherwise possible to query specific properties of the full object.

The properties table includes the following columns:

- `${_mainTable}_id` corresponds to the `id` of the full object in the main table, for example, `manageobjects_id`, or `genericobjects_id`.
- `propkey` is the name of the searchable property, stored in JSON pointer format (for example `/mail`).
- `proptype` is the data type of the property, for example `java.lang.String`. The property type is obtained from the Class associated with the value.
- `propvalue` is the value of property, extracted from the full object that is stored in the main table.

  Regardless of the property data type, this value is stored as a string, so queries against it should treat it as such.

`searchableDefault` *(boolean, optional)*

Specifies whether all properties of the resource should be searchable by default. Properties that are searchable are stored and indexed. You can override the default for individual properties in the `properties` element of the mapping. The preceding example indicates that all properties are searchable, with the exception of the `picture` property.

For large, complex objects, having all properties searchable implies a substantial performance impact. In such a case, a separate insert statement is made in the properties table for each element in the object, every time the object is updated. Also, because these are indexed fields, the recreation of these properties incurs a cost in the maintenance of the index. You should therefore enable `searchable` only for those properties that must be used as part of a WHERE clause in a query.

> **Note**
>
> PostgreSQL repositories do not use the `searchableDefault` property.

`properties`

Lists any individual properties for which the searchable default should be overridden.

Note that if an object was originally created with a subset of `searchable` properties, changing this subset (by adding a new `searchable` property in the configuration, for example) will not cause the existing values to be updated in the properties table for that object. To add the new property to the properties table for that object, you must update or recreate the object.

Improve generic mapping search performance (JDBC)

All properties in a generic mapping are searchable by default. In other words, the value of the `searchableDefault` property is `true` unless you explicitly set it to false. Although there are no individual indexes in a generic mapping, you can improve search performance by setting only those properties that you need to search as `searchable`. Properties that are searchable are created within the corresponding properties table. The properties table exists only for searches or look-ups, and has a composite index, based on the resource, then the property name.

The sample JDBC repository configuration files (`db/database/conf/repo.jdbc.json`) restrict searches to specific properties by setting the `searchableDefault` to `false` for `managed/user` mappings. You must explicitly set `searchable` to true for each property that should be searched. The following sample extract from `repo.jdbc.json` indicates searches restricted to the `userName` property:

```
"genericMapping" : {
    "{managed_user}" : {
        "mainTable" : "manageduserobjects",
        "propertiesTable" : "manageduserobjectproperties",
        "searchableDefault" : false,
        "properties" : {
            "/userName" : {
                "searchable" : true
            }
        }
    }
}
```

With this configuration, IDM creates entries in the properties table only for `userName` properties of managed user objects.

If the global `searchableDefault` is set to false, properties that do not have a searchable attribute explicitly set to true are not written in the properties table.

Explicit mappings (JDBC)

Explicit mapping is more difficult to set up and maintain, but can take complete advantage of the native database facilities.

An explicit table offers better performance and simpler queries. There is less work in the reading and writing of data, because the data is all in a single row of a single table. In addition, it is easier to create different types of indexes that apply to only specific fields in an explicit table. The disadvantage of explicit tables is the additional work required in creating the table in the schema. Also, because rows in a table are inherently more simple, it is more difficult to deal with complex objects. Any non-simple key:value pair in an object associated with an explicit table is converted to a JSON string and stored in the cell in that format. This makes the value difficult to use, from the perspective of a query attempting to search within it.

You can have a generic mapping configuration for most managed objects, *and* an explicit mapping that overrides the default generic mapping in certain cases.

IDM provides a sample configuration, for each JDBC repository, that sets up an explicit mapping for the managed *user* object, and a generic mapping for all other managed objects. This configuration is defined in the files named `/path/to/openidm/db/repository/conf/repo.jdbc-repository-explicit-managed-user.json`. To use this configuration, copy the file that corresponds to your repository to your project's `conf/` directory, and rename it `repo.jdbc.json`. Run the `sample-explicit-managed-user.sql` data definition script (in the `path/to/openidm/db/repository/scripts` directory) to set up the corresponding tables when you configure your JDBC repository.

IDM uses explicit mapping for internal system tables, such as the tables used for auditing.

Depending on the types of usage your system is supporting, you might find that an explicit mapping performs better than a generic mapping. Operations such as sorting and searching (such as those performed in the default UI) tend to be faster with explicitly-mapped objects, for example.

The following sample explicit mapping object illustrates how `internal/user` objects are stored in an explicit table:

```
"explicitMapping" : {
    "internal/user" : {
        "table" : "internaluser",
        "objectToColumn" : {
            "_id" : "objectid",
            "_rev" : { "column" : "rev", "isNotNull" : true },
            "password" : "pwd"
        }
    },
    ...
}
```

`resource-uri` *(string, mandatory)*

Indicates the URI for the resources to which this mapping applies; for example, `internal/user`.

`table` *(string, mandatory)*

The name of the database table in which the object (in this case internal users) is stored.

`objectToColumn` *(string, mandatory)*

The way in which specific managed object properties are mapped to columns in the table.

The mapping can be a simple one to one mapping, for example `"userName": "userName"`, or a more complex JSON map or list. When a column is mapped to a JSON map or list, the syntax is as shown in the following examples:

```
"messageDetail" : { "column" : "messagedetail", "type" : "JSON_MAP" }
```

```
"roles" : { "column" : "roles", "type" : "JSON_LIST" }
```

Available column data types you can specify are `STRING` (the default), `NUMBER`, `JSON_MAP`, `JSON_LIST`, and `FULLOBJECT`.

You can also prevent a column from accepting a `NULL` value, by setting the property `isNotNull` to `true`. This property is optional; if the property is omitted, it will default to `false`. Specifying which columns do not allow a null value can improve performance when sorting and paginating large queries. The syntax is similar to when specifying a column type:

```
"createDate" : { "column" : "createDate", "isNotNull" : true }
```

Caution

Pay particular attention to the following caveats when you map properties to explicit columns in your database:

- Support for data types in columns is restricted to numeric values (`NUMBER`), strings (`STRING`), and boolean values (`BOOLEAN`). Although you can specify other data types, IDM handles all other data types as strings. Your database will need to convert these types from a string to the alternative data type. This conversion is *not guaranteed to work*.
  If the conversion does work, the format might not be the same when the data is read from the database as it was when it was saved. For example, your database might parse a date in the format `12/12/2012` and return the date in the format `2012-12-12` when the property is read.
- Passwords are encrypted before they are stored in the repository. The length of the password column must be long enough to store the encrypted password value, which can vary depending on how it is encrypted and whether it is also hashed.
  The `sample-explicit-managed-user.sql` file referenced in this section sets the password column to a length of 511 characters (`VARCHAR(511)` to account for the additional space an encrypted password requires. For more information about IDM encryption and an example encrypted password value, refer to encrypt and Encode attribute values.
- If your data objects include *virtual properties*, you must include columns in which to store these properties. If you don't explicitly map the virtual properties, you will encounter errors similar to the following when you attempt to create the corresponding object:

  ```
  {
      "code":400,
      "reason":"Bad Request",
      "message":"Unmapped fields [/property-name/0] for type managed/user and table
  openidm.managed_user"
  }
  ```

  To recalculate virtual property values in a query, you must set `executeOnRetrieve` to `true` in the query request parameters. For more information, refer to Property Storage Triggers.

Hybrid mappings (JDBC)

Hybrid mappings are similar to generic mappings, except some object fields are mapped directly to a column, and therefore not stored in the Entity–attribute–value (EAV) properties table. The `fullobject` column still holds all the object data and is used for object constitution. The combination of the explicit field columns and the EAV `properties table` is used for searching.

Object type conversion

You can use the migration service to convert objects from one type to another.

Convert an explicit mapped object to a hybrid mapped object (JDBC)

This procedure demonstrates how to migrate data to a different storage configuration within the same system using the migration service to convert the object data. After you finish the conversion, the converted objects are technically hybrid objects—generically mapped objects that have certain fields that are mapped to explicit columns.

Important

Considerations before you start:

- After you complete the process, object resource paths must stay the same to maintain relationship references.
- You must migrate data to an *empty* table. Unlike generic tables, explicit mapped objects expect the table to contain records from a single object type.
- Changes made to the source object during migration might not be transferred to the new object. To ensure everything is migrated correctly, run the migration during idle time, or when the system is least busy.

This procedure assumes that the repository configuration includes explicitly mapped object types, and that such objects already exist in the corresponding tables. For example:

```
"explicitMapping" : {
...
"managed/objectToConvert" : {
  "table" : "objecttoconvert",
  "objectToColumn" : {
    "_id" : "objectid",
    "_rev" : "rev",
    "desc" : "descr"
  }
}
```

1. Create the new generic table and associated properties table. Adjust the following example to match your repository requirements, as needed:

```
CREATE TABLE `openidm`.`objecttoconvert_gen` (
`id` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT ,
`objecttypes_id` BIGINT UNSIGNED NOT NULL ,
`objectid` VARCHAR(255) NOT NULL ,
`rev` VARCHAR(38) NOT NULL ,
`descr` VARCHAR(255) NOT NULL ,
`fullobject` MEDIUMTEXT NULL ,
PRIMARY KEY (`id`) ,
UNIQUE INDEX `idx-objecttoconvert_object` (`objecttypes_id` ASC, `objectid` ASC) ,
INDEX `fk_objecttoconvert_objectypes` (`objecttypes_id` ASC) ,
CONSTRAINT `fk_objecttoconvert_objectypes`
    FOREIGN KEY (`objecttypes_id` )
        REFERENCES `openidm`.`objecttypes` (`id` )
        ON DELETE CASCADE
        ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `openidm`.`objecttoconvert_genproperties` (
`objecttoconvert_gen_id` BIGINT UNSIGNED NOT NULL ,
`propkey` VARCHAR(255) NOT NULL ,
`proptype` VARCHAR(32) NULL ,
`propvalue` VARCHAR(2000) NULL ,
`propindex` BIGINT NOT NULL DEFAULT 0,
PRIMARY KEY (`objecttoconvert_gen_id`, `propkey`, `propindex`),
INDEX `fk_objecttoconvertproperties_managedobjects` (`objecttoconvert_gen_id` ASC) ,
INDEX `idx_objecttoconvertproperties_propkey` (`propkey` ASC) ,
INDEX `idx_objecttoconvertproperties_propvalue` (`propvalue`(255) ASC) ,
CONSTRAINT `fk_objecttoconvertproperties_objecttoconvert`
FOREIGN KEY (`objecttoconvert_gen_id` )
REFERENCES `openidm`.`objecttoconvert_gen` (`id` )
ON DELETE CASCADE
ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

2. Modify `conf/repo.jdbc.json` to map the object path in the generic mapping section to the empty generic table. If the migrated data will have additional searchable columns, add them now.
3. Create a `conf/migration.json` file with the following details:
    - Update the authentication settings to match the system configuration.
    - Modify the `instanceUrl` to point to the same system.

      For example:

```
{
  "enabled" : true,
  "endpoint" : "",
  "connection" : {
    "instanceUrl" : "http://localhost:8080/openidm/",
    "authType" : "basic",
    "userName" : "openidm-admin",
    "password" : "openidm-admin"
  },
  "mappings" : [
    {
      "target" : "repo/managed/objectToConvert_gen",
      "source" : "repo/managed/objectToConvert"
    }
  ]
}
```

4. Call the migration service to view the mapping name that was generated:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header 'Accept-API-Version: resource=1.0' \
--request POST 'http://localhost:8080/openidm/migration?_action=mappingNames'
[
  [
    "repoManagedObjecttoconvert_repoManagedObjecttoconvertGen"
  ]
]
```

5. Start the migration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/migration?
_action=migrate&mapping=repoManagedObjecttoconvert_repoManagedObjecttoconvertGen"
{
  "migrationResults": {
    "recons": [
      {
        "name": "repoManagedObjecttoconvert_repoManagedObjecttoconvertGen",
        "status": "PENDING"
      }
    ]
  }
}
```

6. You must wait until the migration is completed. To check the status of the migration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header 'Accept-API-Version: resource=1.0' \
--request POST 'http://localhost:8080/openidm/migration?_action=status'
```

```
{
  "migrationResults": {
    "recons": [
      {
        "name": "repoManagedObjecttoconvert_repoManagedObjecttoconvertGen",
        "status": {
          "_id": "820a1c66-6f1a-41d8-82a4-fc5a2d246326-424",
          "mapping": "repoManagedObjecttoconvert_repoManagedObjecttoconvertGen",
          "state": "SUCCESS",
          "stage": "COMPLETED_SUCCESS",
          "stageDescription": "reconciliation completed.",
          "progress": {
            "source": {
              "existing": {
                "processed": 0,
                "total": "9"
              }
            },
            "target": {
              "existing": {
                "processed": 0,
                "total": "?"
              },
              "created": 0,
              "unchanged": 0,
              "updated": 0,
              "deleted": 0
            },
            "links": {
              "existing": {
                "processed": 0,
                "total": "0"
              },
              "created": 0
            }
          },
          "situationSummary": {
            "SOURCE_IGNORED": 0,
            "FOUND_ALREADY_LINKED": 0,
            "UNQUALIFIED": 0,
            "ABSENT": 0,
            "TARGET_IGNORED": 0,
            "MISSING": 0,
            "ALL_GONE": 0,
            "UNASSIGNED": 0,
            "AMBIGUOUS": 0,
            "CONFIRMED": 0,
            "LINK_ONLY": 0,
            "SOURCE_MISSING": 0,
            "FOUND": 0
          },
          "statusSummary": {
            "SUCCESS": 0,
            "FAILURE": 9
          },
          "durationSummary": {
            "sourceObjectQuery": {
              "min": 26,
              "max": 33,
              "mean": 30,
```

```
        "count": 9,
        "sum": 277,
        "stdDev": 2
      },
      "sourceQuery": {
        "min": 37,
        "max": 37,
        "mean": 37,
        "count": 1,
        "sum": 37,
        "stdDev": 0
      },
      "auditLog": {
        "min": 0,
        "max": 1,
        "mean": 0,
        "count": 11,
        "sum": 9,
        "stdDev": 0
      },
      "linkQuery": {
        "min": 4,
        "max": 4,
        "mean": 4,
        "count": 1,
        "sum": 4,
        "stdDev": 0
      },
      "correlationQuery": {
        "min": 8,
        "max": 18,
        "mean": 15,
        "count": 9,
        "sum": 139,
        "stdDev": 4
      },
      "sourcePhase": {
        "min": 113,
        "max": 113,
        "mean": 113,
        "count": 1,
        "sum": 113,
        "stdDev": 0
      }
    },
    "parameters": {
      "sourceQuery": {
        "resourceName": "external/migration/repo/managed/objectToConvert",
        "queryFilter": "true",
        "_fields": "_id"
      },
      "targetQuery": {
        "resourceName": "repo/managed/objectToConvert_gen",
        "queryFilter": "true",
        "_fields": "_id"
      }
    },
    "started": "2021-01-20T18:22:34.026Z",
    "ended": "2021-01-20T18:22:34.403Z",
    "duration": 377,
    "sourceProcessedByNode": {}
```

```
        }
      }
    ]
  }
}
```

> **Note**
>
> Optionally, you can run the migration again to account for changes that may have occurred during the original migration.

The data is now migrated to the new tables, but IDM is still referencing the previous mapping.

7. Edit the `repo.jdbc.json` file:
   - Remove the old mapping from `explicitMapping`:

```
"explicitMapping" : {
...
"managed/objectToConvert" : {
  "table" : "objecttoconvert",
  "objectToColumn" : {
    "_id" : "objectid",
    "_rev" : "rev",
    "desc" : "descr"
  }
}
```

   - Modify the newly added `genericMapping` to point to the old resource path:

```
"genericMapping" : {
  ...
  "managed/objectToConvert" : {
    "mainTable" : "objecttoconvert_gen",
    "propertiesTable" : "objecttoconvert_genproperties",
    "searchableDefault" : false,
    "objectToColumn" : {
      "_id" : "objectid",
      "_rev" : "rev",
      "desc" : "descr"
    },
    "properties": {
      "/stringArrayField" : {
        "searchable" : true
      }
    }
  },
}
```

8. Run a SQL update statement so that the `objecttypes` table points the temporary object type to the original object type. Adjust the following example to match your repository requirements, as needed:

```
update openidm.objecttypes set objecttype = 'managed/objectToConvert' where objecttype = 'managed/
objectToConvert_gen';
```

Convert a generic mapped object to an explicit mapped object (JDBC)

This procedure demonstrates how to migrate data to a different storage configuration within the same system using the migration service to convert the object data.

> **Important**
>
> Considerations before you start:
>
> - After you complete the process, object resource paths must stay the same to maintain relationship references.
> - You must migrate data to an *empty* table. Unlike generic tables, explicit mapped objects expect the table to contain records from a single object type.
> - During the migration, changes made to the source object might not be transferred to the new object. To ensure everything is migrated correctly, run the migration during idle time, or when the system is least busy.

This procedure assumes an existing generic object resource path of `managed/objectToConvert`, with objects stored in the generic objects table `genericobjects`. A sample object might be:

```
{
  "_id" : "4213-2134-23423",
  "_rev" : "AB231A",
  "name" : "Living room camera",
  "properties": { "location" : "45.123N100.123W", "uptime" : 123123 },
  "otherProperties" : { "bla": "blabla", "blahdee" : "da"}
}
```

Before you start, consider the following:

- Make sure to map a column to each field of your object.
- Fields that are objects, not simple scalar values, will be stored as serialized JSON, and won't be easily searchable.
- Object instances are constituted by selecting the mapped columns and putting the data in the JSON object using the field path that the column is mapped to.
- Create table indexes that are inline with your system's usage of searches and sorting of the column data. For example, modify or add indexes to include all newly created columns for any fields that were configured as searchable.

1. Create the new explicit table:

```
CREATE TABLE `openidm`.`objectToConvert` (
  `objectid` VARCHAR(255) NOT NULL ,
  `rev` VARCHAR(38) NOT NULL ,
  `name` VARCHAR(255) NOT NULL ,
  `location` VARCHAR(38) NULL ,
  `uptime` BIGINT NULL ,
  `misc` MEDIUMTEXT NULL,
PRIMARY KEY (`objectid`));
```

2. Modify `conf/repo.jdbc.json` to add a new mapping for the object type in the `explicitMapping` node. To avoid conflict with the generically mapped object path, slightly modify the resource path. A new explicit mapping example:

```
"explicitMapping" : {
  ...
    "managed/objectToConvert_explicit": {
      "table": "objectToConvert",
      "objectToColumn" : {
        "_id" : "objectid",
        "_rev" : { "column" : "rev", "isNotNull" : true },
        "name" : { "column" : "name", "isNotNull" : true },
        "properties/location" : { "column": "location" },
        "properties/uptime" : { "column" : "uptime" },
        "otherProperties" : {
          "column" : "misc",
          "type" : "JSON_MAP"
        }
      }
    }
    ...
  ...
}
```

3. Create a `conf/migration.json` file with the following details:
   - Update the authentication settings to match the system configuration.
   - Modify the `instanceUrl` to point to the same system.

     For example:

```
{
  "enabled" : true,
  "endpoint" : "",
  "connection" : {
    "instanceUrl" : "http://localhost:8080/openidm/",
    "authType" : "basic",
    "userName" : "openidm-admin",
    "password" : "openidm-admin"
  },
  "mappings" : [
    {
      "target" : "repo/managed/objectToConvert_explicit",
      "source" : "repo/managed/objectToConvert"
    }
  ]
}
```

4. Call the migration service to view the mapping name that was generated:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header 'Accept-API-Version: resource=1.0' \
--request POST 'http://localhost:8080/openidm/migration?_action=mappingNames'
```

IDM returns something similar to:

```
[
  [
    "repoManagedObjecttoconvert_repoManagedObjecttoconvertExplicit"
  ]
]
```

5. Start the migration:

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

IDM returns something similar to:

```
{
  "migrationResults": {
    "recons": [
      {
        "name": "repoManagedObjecttoconvert_repoManagedObjecttoconvertExplicit",
        "status": "PENDING"
      }
    ]
  }
}
```

6. You must wait until the migration completes. To check the status of the migration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header 'Accept-API-Version: resource=1.0' \
--request POST 'http://localhost:8080/openidm/migration?_action=status'
```

IDM returns something similar to:

```
{
  "migrationResults": {
    "recons": [
      {
        "name": "repoManagedObjecttoconvert_repoManagedObjecttoconvertExplicit",
        "status": {
          "_id": "820a1c66-6f1a-41d8-82a4-fc5a2d246326-424",
          "mapping": "repoManagedObjecttoconvert_repoManagedObjecttoconvertExplicit",
          "state": "SUCCESS",
          "stage": "COMPLETED_SUCCESS",
          "stageDescription": "reconciliation completed.",
          "progress": {
            "source": {
              "existing": {
                "processed": 0,
                "total": "9"
              }
            },
            "target": {
              "existing": {
                "processed": 0,
                "total": "?"
              },
              "created": 0,
              "unchanged": 0,
              "updated": 0,
              "deleted": 0
            },
            "links": {
              "existing": {
                "processed": 0,
                "total": "0"
              },
              "created": 0
            }
          },
          "situationSummary": {
            "SOURCE_IGNORED": 0,
            "FOUND_ALREADY_LINKED": 0,
            "UNQUALIFIED": 0,
            "ABSENT": 0,
            "TARGET_IGNORED": 0,
            "MISSING": 0,
            "ALL_GONE": 0,
            "UNASSIGNED": 0,
            "AMBIGUOUS": 0,
            "CONFIRMED": 0,
            "LINK_ONLY": 0,
            "SOURCE_MISSING": 0,
            "FOUND": 0
          },
          "statusSummary": {
            "SUCCESS": 0,
            "FAILURE": 9
          },
          "durationSummary": {
            "sourceObjectQuery": {
              "min": 26,
              "max": 33,
              "mean": 30,
```

```
      "count": 9,
      "sum": 277,
      "stdDev": 2
    },
    "sourceQuery": {
      "min": 37,
      "max": 37,
      "mean": 37,
      "count": 1,
      "sum": 37,
      "stdDev": 0
    },
    "auditLog": {
      "min": 0,
      "max": 1,
      "mean": 0,
      "count": 11,
      "sum": 9,
      "stdDev": 0
    },
    "linkQuery": {
      "min": 4,
      "max": 4,
      "mean": 4,
      "count": 1,
      "sum": 4,
      "stdDev": 0
    },
    "correlationQuery": {
      "min": 8,
      "max": 18,
      "mean": 15,
      "count": 9,
      "sum": 139,
      "stdDev": 4
    },
    "sourcePhase": {
      "min": 113,
      "max": 113,
      "mean": 113,
      "count": 1,
      "sum": 113,
      "stdDev": 0
    }
  },
  "parameters": {
    "sourceQuery": {
      "resourceName": "external/migration/repo/managed/objectToConvert",
      "queryFilter": "true",
      "_fields": "_id"
    },
    "targetQuery": {
      "resourceName": "repo/managed/objectToConvert_explicit",
      "queryFilter": "true",
      "_fields": "_id"
    }
  },
  "started": "2021-01-20T18:22:34.026Z",
  "ended": "2021-01-20T18:22:34.403Z",
  "duration": 377,
  "sourceProcessedByNode": {}
```

```
        }
      }
    ]
  }
}
```

> **Note**
>
> Optionally, you can run the migration again to account for changes that may have occurred during the original migration.

The data is now migrated to the new tables, but IDM is still referencing the previous mapping and generic table.

7. Edit the `repo.jdbc.json` file:
   - If the mapping of the generic resource had a mapping, it should be removed. If the generic resource was included in the `managed/*` path, as in the example, there is nothing to remove.
   - Modify the object path from `managed/objectToConvert_explicit` to `managed/objectToConvert`.
8. Save the `repo.jdbc.json` file.

   Until IDM processes the configuration change, REST requests are unavailable.
9. Once IDM finishes processing the configuration change, it is safe to delete the object data from the original generic table. Using a proper *delete cascade*, the searchable properties of the generic object are automatically deleted from the generic properties table. For example:

```
delete
from
    managedobjects
where
    objecttypes_id = (
    select
        o2.id
    from
        objecttypes o2
    where
        objecttype = "managed/objectToConvert");
```

Convert a generic mapped object to a hybrid mapped object (JDBC)

This procedure demonstrates how to convert a generically stored property to an explicitly stored property; moving property value storage out of the object's property table, and into a new column on the object table. After you finish the conversion, the converted objects are technically hybrid objects—generically mapped objects that have certain fields that are mapped to explicit columns.

Important

Considerations before you start:

- Changes made to the source object during migration might not be transferred to the new object. To ensure everything is migrated correctly, run the migration during idle time, or when the system is least busy.

This procedure assumes an existing generic object resource path as: `managed/objectToConvert`, with objects stored in the generic objects table `objecttoconvertobjects`. A sample object might be:

```
{
  "_id" : "4213-2134-23423",
  "_rev" : "AB231A",
  "name" : "Living room camera",
  "properties": { "location" : "45.123N100.123W", "uptime" : 123123 },
  "otherProperties" : { "bla": "blabla", "blahdee" : "da"}
}
```

1. Create a new database column for the explicit field data:

   ```
   alter table openidm.objecttoconvertobjects add column `name` varchar(255);
   ```

2. Edit the `genericMapping` section of the `conf/repo.jdbc.json` file:

   - For each object to convert, add the `objectToColumn` configuration for the fields to explicitly map to a column. For example:

     ```
     "genericMapping" : {
       ...
         "managed/objectToConvert": {
           "mainTable" : "objecttoconvertobjects",
           "propertiesTable" : "objecttoconvertobjectsproperties",
           "searchableDefault" : true,
           "objectToColumn" : {
             "name" : "name"
           }
         }
         ...
     }
     ```

   - If the object is defined with a wildcard mapping, such as `managed/*`, create a new mapping specifically for the object conversion.

3. Save the `conf/repo.jdbc.json` file.

   Until IDM processes the configuration change, REST requests are unavailable.

4. For any added columns to be usable, they must be reindexed and populated.

The `rewriteObjects.js` script can read and rewrite object data to match the config from the `conf/repo.jdbc.json` file. The script reads one page of data at a time, and then writes out each object, one at a time. Consider the page size and queryFilter to efficiently process the data by splitting the data into groups that can run in parallel. The request will not return until *ALL* pages have been processed.

For example, to run the `rewriteObjects.js` script with 1000 objects per page:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header 'Accept-API-Version: resource=1.0' \
--header 'content-type: application/json' \
--request POST \
--data-raw '{
  "type":"text/javascript",
  "file":"bin/defaults/script/update/rewriteObjects.js",
  "globals" : {
    "rewriteConfig" :{
      "queryFilter": "true",
      "pageSize": 1000,
      "objectPaths": [
        "repo/managed/objectToConvert"
      ]
    }
  }
}' \
"http://localhost:8080/openidm/script?_action=eval"
```

5. Now that the new column contains data, edit or create new indexes so that the new column can be queried efficiently. For example:

```
alter table openidm.objecttoconvertobjects add index `idx_obj_name` (`name`);
```

Mappings with a DS repository

For both generic and explicit mappings, IDM maps object types using a `dnTemplate` property. The `dnTemplate` is effectively a pointer to where the object is stored in DS. For example, the following excerpt of the default `repo.ds.json` file shows how configuration objects are stored under the DN `ou=config,dc=openidm,dc=forgerock,dc=com`:

```
"config": {
    "dnTemplate": "ou=config,dc=openidm,dc=forgerock,dc=com"
}
```

Generic mappings (DS)

By default, IDM uses a generic mapping for all objects *except* the following:

- Internal users, roles, and privileges
- Links

- Clustered reconciliation target IDs

> **Note**
>
> Clustered reconciliation is not supported with a DS repository.

- Locks
- Objects related to queued synchronization

With a generic mapping, all the properties of an object are stored as a single JSON blob in the `fr-idm-json` attribute. To create a new generic mapping, you need only specify the `dnTemplate`; that is, where the object will be stored in the directory tree.

You can specify a wildcard mapping, that stores all nested URIs under a particular branch of the directory tree, for example:

```
"managed/*": {
    "dnTemplate": "ou=managed,dc=openidm,dc=forgerock,dc=com"
}
```

With this mapping, all objects under `managed/`, such as `managed/user` and `managed/device`, will be stored in the branch `ou=managed,dc=openidm,dc=forgerock,dc=com`. You do not have to specify separate mappings for each of these objects. The mapping creates a new `ou` for each object. So, for example, `managed/user` objects will be stored under the DN `ou=user,ou=managed,dc=openidm,dc=forgerock,dc=com` and `managed/device` objects will be stored under the DN `ou=device,ou=managed,dc=openidm,dc=forgerock,dc=com`.

In cases where you want to improve external DS search performance, you can configure indexing for your resources within DS. For more information on indexing in DS, refer to Indexes in the DS documentation.

Explicit mappings (DS)

The default configuration uses a generic mapping for managed user objects. To use an explicit mapping for managed user objects, change the repository configuration *before you start IDM for the first time*.

To set up an explicit mapping:

1. Copy the `repo.ds-explicit-managed-user.json` file to your project's `conf` directory, and rename that file `repo.ds.json`:

```
cp /path/to/openidm/db/ds/conf/repo.ds-explicit-managed-user.json project-dir/conf/repo.ds.json
```

> **Important**
>
> This file is configured for an embedded DS repository by default. To set up an explicit mapping for an external DS repository, change the value of the `embedded` property to `false` and add the following properties:

```
"security": {
  "trustManager": "file",
  "fileBasedTrustManagerType": "JKS",
  "fileBasedTrustManagerFile": "&{idm.install.dir}/security/truststore",
  "fileBasedTrustManagerPasswordFile": "&{idm.install.dir}/security/storepass"
},
"ldapConnectionFactories": {
  "bind": {
    "connectionSecurity": "startTLS",
    "heartBeatIntervalSeconds": 60,
    "heartBeatTimeoutMilliSeconds": 10000,
    "primaryLdapServers": [
      {
        "hostname": "localhost",
        "port": 31389
      }
    ],
    "secondaryLdapServers": []
  },
  "root": {
    "inheritFrom": "bind",
    "authentication": {
      "simple": {
        "bindDn": "uid=admin",
        "bindPassword": "password"
      }
    }
  }
}
```

For more information on these properties, refer to DS Repository Configuration.

2. Start IDM.

IDM uses the DS REST to LDAP gateway to map JSON objects to LDAP objects stored in the directory. To create additional explicit mappings, you must specify the LDAP `objectClasses` to which the object is mapped, and how each property maps to its corresponding LDAP attributes. Specify at least the property `type` and the corresponding `ldapAttribute`. For relationships between objects, you must explicitly define those objects in the repository configuration.

The following excerpt shows an example of an explicit managed user object mapping:

```
"managed/user" : {
    "dnTemplate": "ou=user,ou=managed,dc=openidm,dc=forgerock,dc=com",
    "objectClasses": [
        "person",
        "organizationalPerson",
        "inetOrgPerson",
        "fr-idm-managed-user-explicit",
        "inetuser"
    ],
    "properties": {
        "_id": {
            "type": "simple", "ldapAttribute": "uid", "isRequired": true, "writability": "createOnly"
        },
        "userName": {
            "type": "simple", "ldapAttribute": "cn"
        },
        "password": {
            "type": "json", "ldapAttribute": "fr-idm-password"
        },
        "accountStatus": {
            "type": "simple", "ldapAttribute": "fr-idm-accountStatus"
        },
        "roles": {
            "type": "json", "ldapAttribute": "fr-idm-role", "isMultiValued": true
        },
        "effectiveRoles": {
            "type": "json", "ldapAttribute": "fr-idm-effectiveRole", "isMultiValued": true
        },
        "effectiveAssignments": {
            "type": "json", "ldapAttribute": "fr-idm-effectiveAssignment", "isMultiValued": true
        },
        ...
    }
}
```

You do not need to map the `_rev` (revision) property of an object as this property is implicit in all objects and maps to the DS `etag` operational attribute.

If your data objects include *virtual properties*, you must include property mappings for these properties. If you don't explicitly map the virtual properties, you will encounter errors similar to the following when you attempt to create the corresponding object:

```
{
    "code": 400,
    "reason": "Bad Request",
    "message": "Unmapped fields..."
}
```

For more information about the REST to LDAP property mappings, refer to Mapping Configuration in the *DS REST API Guide*.

For performance reasons, the DS repository does not apply unique constraints to links. This behavior is different to the JDBC repositories, where uniqueness on link objects is enforced.

Important

DS currently has a default index entry limit of 4000. Therefore, you cannot query more than 4000 records unless you create a Virtual List View (VLV) index. A VLV index is designed to help DS respond to client applications that need to browse through a long list of objects.
You cannot create a VLV index on a JSON attribute. For generic mappings, IDM avoids this restriction by using client-side sorting and searching. However, for explicit mappings you *must* create a VLV index for any filtered or sorted results, such as results displayed in a UI grid. To configure a VLV index, use the `dsconfig` command described in Virtual List View Index in the *DS Configuration Guide*.

Specify how IDM IDs map to LDAP entry names

The DS REST2LDAP configuration lets you set a `namingStrategy` that specifies how LDAP entry names are mapped to JSON resources. When IDM stores its objects in a DS repository, this `namingStrategy` determines how the IDM `_id` value maps to the Relative Distinguished Name (RDN) of the corresponding DS object.

The `namingStrategy` is specified as part of the `explicitMapping` of an object in the `repo.ds.json` file. The following example shows a naming strategy configuration for an explicit managed user mapping:

```
"resourceMapping": {
    "defaultMapping": {
        "dnTemplate": "ou=generic,dc=openidm,dc=forgerock,dc=com"
    },
    ...
    "explicitMapping": {
        "managed/user": {
            "dnTemplate": "ou=user,ou=managed,dc=openidm,dc=forgerock,dc=com",
            "objectClasses": [
                "person",
                "organizationalPerson",
                "inetOrgPerson",
                "fr-idm-managed-user-explicit"
            ],
            "namingStrategy": {
                "type": "clientDnNaming",
                "dnAttribute": "uid"
            },
            ...
        }
    }
}
```

The `namingStrategy` can be one of the following:

- `clientDnNaming` - IDM provides an `_id` to DS and that `_id` is used to generate the DS RDN. In the following example, the IDM `_id` maps to the LDAP `uid` attribute:

```
{
    "namingStrategy": {
        "type": "clientDnNaming",
        "dnAttribute": "uid"
    }
}
```

With this *default* configuration, entries are stored in DS with a DN similar to the following:

```
"uid=idm-uuid,ou=user,ou=managed,dc=openidm,dc=forgerock,dc=com"
```

> **Note**
>
> If these default DNs are suitable in your deployment, you do not have to change anything with regard to the naming strategy.

- `clientNaming` - IDM provides an `_id` to DS but the DS RDN is derived from a different user attribute in the LDAP entry. In the following example, the RDN is the `cn` attribute. The `_id` that IDM provides for the object maps to the LDAP `uid` attribute:

```
{
    "namingStrategy": {
        "type": "clientNaming",
        "dnAttribute": "cn",
        "idAttribute": "uid"
    }
}
```

With this configuration, entries are stored in DS with a DN similar to the following:

```
"cn=username,ou=user,ou=managed,dc=openidm,dc=forgerock,dc=com"
```

Specifying a `namingStrategy` is optional. If you do not specify a strategy, the default is `clientDnNaming` with the following configuration:

```
{
    "namingStrategy" : {
        "type" : "clientDnNaming",
        "dnAttribute" : "uid"
    },
    "properties: : {
        "_id": {
            "type": "simple",
            "ldapAttribute": "uid",
            "isRequired": true,
            "writability": "createOnly"
        },
        ...
    }
}
```

Note

If you do not set a `dnAttribute` as part of the naming strategy, the value of the `dnAttribute` is taken from the value of the `ldapAttribute` on the `_id` property.

Relationship properties in a DS repository

The IDM object model lets you define relationships between objects. In a DS repository, relationships are implemented using the `reference` and `reverseReference` REST to LDAP property types. For more information about the `reference` and `reverseReference` property types, read the JSON property mapping section of the *DS HTTP User Guide*.

Relationship properties must be defined in the repository configuration (`repo.ds.json`), for both generic and explicit object mappings.

The following property definitions for a `managed/user` object show how the relationship between a `manager` and their `reports` is defined in the repository configuration:

```
"managed/user" : {
    "dnTemplate" : "ou=user,ou=managed,dc=openidm,dc=forgerock,dc=com",
    ...
    "properties" : {
        ...
        "reports" : {
            "type" : "reverseReference",
            "resourcePath" : "managed/user",
            "propertyName" : "manager",
            "isMultiValued" : true
        },
        "manager" : {
            "type" : "reference",
            "ldapAttribute" : "fr-idm-managed-user-manager",
            "primaryKey" : "uid",
            "resourcePath" : "managed/user",
            "isMultiValued" : false
        },
        ...
    }
}
```

This configuration sets the `reports` property as a `reverseReference`, or reverse *relationship* of the `manager` property. This means that if you add a `manager` to a user, the user automatically becomes one of the `reports` of that manager.

Note the `ldapAttribute` defined in the relationship object (`fr-idm-managed-user-manager` in this case). Your DS schema must include this attribute, and an object class that contains this attribute. Relationship attributes in the DS schema must use the Name and JSON with id syntax.

The following example shows the DS schema definition for the IDM `manager` property:

```
attributeTypes: ( 1.3.6.1.4.1.36733.2.3.1.69
  NAME 'fr-idm-managed-user-manager'
  DESC 'Reference to a users manager'
  SINGLE-VALUE
  SYNTAX 1.3.6.1.4.1.36733.2.1.3.12
  EQUALITY nameAndOptionalCaseIgnoreJsonIdEqualityMatch
  X-STABILITY 'Internal' )
```

Important

If you define a relationship in the managed object configuration and you do not define that relationship as a reference or reverse reference in the repository configuration (`repo.ds.json`), you will be able to query the relationships, but filtering and sorting on those queries will not work. This is the case when you define relationship objects in the admin UI—the relationship is defined only in the managed object configuration and not in the repository configuration.
In this case, queries such as the following are not supported:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/_id/managedOrgs?
_pageSize=50&_sortKeys=_id&_totalPagedResultsPolicy=ESTIMATE&_queryFilter=true"
```

This restriction includes delegated admin privilege filters.

Access Data Objects

Access data objects using scripts

IDM's uniform programming model means that all objects are queried and manipulated in the same way, using the Resource API. The URL or URI that is used to identify the target object for an operation depends on the object type. For more information about scripts and the objects available to scripts, refer to Scripting.

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

```
val = openidm.read("managed/organization/mysampleorg")
val = openidm.read("system/mysystem/account")
val = openidm.read("config/custom/mylookuptable")
val = openidm.read("repo/custom/mylookuptable")
```

For information about constructing an object ID, refer to URI Scheme.

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

```
openidm.update("managed/organization/mysampleorg", rev, object)
openidm.update("system/mysystem/account", rev, object)
```

You can apply a partial update to a managed or system object by using the `patch()` function:

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

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

Access data objects using the REST API

IDM provides RESTful access to data objects through the ForgeRock Common REST API. To access objects over REST, you can use a browser-based REST client, such as the *Simple REST Client* for Chrome, or *RESTClient* for Firefox. Alternatively, you can use the curl command-line utility.

For a comprehensive overview of the REST API, refer to the REST API reference.

To obtain a managed object through the REST API, depending on your security settings and authentication configuration, perform an HTTP GET on the corresponding URL, for example `http://localhost:8080/openidm/managed/organization/mysampleorg`.

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

In general, you can map any HTTP request to the corresponding `openidm.method` call. The following example shows how the parameters provided in an `openidm.query` request correspond with the key-value pairs that you would include in a similar HTTP GET request:

Reading an object using the Resource API:

```
openidm.query("managed/user", { "_queryFilter": "true" }, ["userName","sn"])
```

Reading an object using the REST API:

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

Access data objects by remote proxy

You can proxy REST requests to a remote IDM or Identity Cloud instance using the `openidm/external/idm/instanceName` endpoint.

> **Important**
>
> For more information on determining the exact value for `instanceName`, refer to How to determine the value for `instanceName`.

This functionality lets you treat any other IDM or Identity Cloud instance as a resource within the one you are managing. You can then use it in a sync mapping, call actions on it, use it within scripts, or use it in any other way that you might use a resource in IDM. You can call any endpoint in the remote IDM system using this proxy.

A few situations where this feature may be useful include:

- Situations where some, but not all, data needs to be migrated from an older version to a newer release.
- Situations where a development or testing environment has data that needs to be synced into the production environment.
- Situations where data is deployed in geographically diverse data centers and changes need to be kept in sync with one another.
- Situations where a new instance needs to sync data between existing on-premises and cloud instances.

This feature does not support liveSync/implicit sync from the remote IDM resources. This means that you will be limited to using recon when it comes to pulling data from a remote system.

> **Note**
>
> If requests sent to the source server include an `X-Requested-With` header, the value of the header will be set to `RemoteIDMProxy`.

How to determine the value for `instanceName`

The `instanceName` is a fragment of the external configuration's name. You can determine the value for `instanceName` using REST or the filesystem:

Using the filesystem

1. Go to `/path/to/openidm/conf/`.
2. Locate the file named `external.idm-instanceName.json`.

For example, a file named `external.idm-name1.json` would be available as a remote system at the `openidm/external/idm/name1` endpoint.

Using REST

1. Get the configurations:

Request

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"https://localhost:8443/openidm/config/"
```

2. Locate the external configuration:

Return

```
{
  "_id": "",
  "configurations": [
    ...
    {
      "_id": "emailTemplate/welcome",
      "pid": "emailTemplate.212e...f7a",
      "factoryPid": "emailTemplate"
    },
    ...
    {
      "_id": "external.idm/name1", (1)
      "pid": "external.idm.29cd...f4a",
      "factoryPid": "external.idm"
    },
    ...
  ]
}
```

1 In this example, the external configuration `"_id": "external.idm/name1"` would be available as a remote system at the `openidm/external/idm/name1` endpoint.

Prerequisites

To connect to a remote instance over SSL, you must import the remote instance's server certificate into your local instance's truststore. For example:

```
keytool \
-import \
-alias fr-platform \
-keystore security/truststore \
-file ~/fr-platform.pem
```

Mapping

To use the remote IDM proxy in a synchronization mapping, add the following to your `sync.json` file or individual mapping file (updating the values as necessary):

```
{
  "name" : "onprem_user_to_fidc_alpha_user",
  "source" : "external/idm/65/managed/user",
  "target" : "external/idm/fidc/managed/alpha_user"
}
```

Authentication

Authentication against the remote IDM instance is supported via `basic` authentication, or `bearer` token authentication when IDM is configured to use rsFilter. The authentication strategy determines some of the parameters required for the request.

Property	Required?	Definition
enabled	No	The enable state of the service. Default is `true`.
scope	No	The requested OAuth2 scope(s).
scopeDelimiter	No	The scope delimiter to use. Defaults to space.
authtype	Yes	The authentication strategy to use. Either `basic` or `bearer`.
instanceUrl	Yes	The URL of the remote instance to relay the request to.
userName	With `basic` auth	The basic authentication user name.
password	With `basic` auth	The basic authentication password.
clientId	With `bearer` auth	The clientId used to request an access token from the token endpoint.
clientSecret	With `bearer` auth	The client secret used to request an access token from the token endpoint.
tokenEndpoint	With `bearer` auth	The OAuth2 token endpoint.

Examples

Basic authentication

```
{
  "enabled" : true,
  "authType" : "basic",
  "instanceUrl" : "https://localhost:8443/openidm/",
  "userName" : "openidm-admin",
  "password" : "openidm-admin"
}
```

Bearer/Oauth2 authentication

```
{
  "enabled" : true,
  "authType" : "bearer",
  "instanceUrl" : "https://fr-platform.iam.example.com/openidm/",
  "clientId" : "idm-provisioning",
  "clientSecret" : "password",
  "scope" : [ ],
  "tokenEndpoint" : "https://fr-platform.iam.example.com/am/oauth2/realms/root/access_token",
  "scopeDelimiter" : " "
}
```

REST request

Request

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--insecure \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'https://localhost:8443/openidm/external/idm/platform/managed/user?_queryFilter=true'
```

Return


```
{
  "result": [
    {
      "_id": "95b2b43c-621e-4bca-8a97-efc768f17751",
      "_rev": "00000000f20217df",
      "userName": "testUser",
      "accountStatus": "active",
      "givenName": "Test",
      "sn": "User",
      "mail": "testUser@test.com"
    }
  ],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```


Script

```
openidm.query("external/idm/fidc/managed/alpha_user", {"_queryFilter": "userName eq 'bjensen'"});
```

Define and call data queries

An advanced query model enables you to define queries and to call them over the REST or Resource API. The following types of queries are supported, on both managed, and system objects:

- Common filter expressions
- Parameterized, or predefined queries
- Native query expressions

> **Tip**
>
> For limits on queries in progressive profiling, refer to Custom Progressive Profile Conditions.

Queries on object array properties (JDBC)

Support for queries on object array properties requires the following:

- A JDBC repository with generic object mapping. Queries on arrays are not supported with explicit mappings. If you need to convert from explicitly mapped objects to generic, refer to Convert an Explicit Mapped Object to a Hybrid Mapped Object (JDBC).
- For PostgreSQL only, you must configure array fields. Additional information about PostgreSQL JSON functions.

- For JDBC repositories other than PostgreSQL, the array property must be configured as searchable. If you add additional properties as searchable after the initial install/migration of IDM, run the `/path/to/openidm/bin/defaults/script/update/rewriteObjects.js` script, specifying the new `objectPaths` of properties to make searchable:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header 'X-OpenIDM-NoSession: true' \
--request POST \
--data-raw '{
  "type": "text/javascript",
  "file": "/path/to/openidm/bin/defaults/script/update/rewriteObjects.js",
  "globals": {
    "rewriteConfig": {
      "queryFilter": "true",
      "pageSize": 1000,
      "objectPaths": [
        "repo/config",
        "repo/internal/usermeta",
        "repo/managed/role",
        "repo/managed/user",
        "repo/reconprogressstate",
        "repo/relationships",
        "repo/scheduler/triggers"
      ]
    }
  }
}' \
"http://localhost:8080/openidm/script/?_action=eval"
```

- Do not use array fields in a `sortKey`.

Special characters in queries

JavaScript query invocations are not subject to the same URL-encoding requirements as GET requests. Because JavaScript supports the use of single quotes, it is not necessary to escape the double quotes from most examples in this guide. Make sure to protect against pulling in data that could contain special characters, such as double-quotes (`"`). The following example shows one method of handling special characters:

```
"correlationQuery" : {
  "type" : "text/javascript",
  "source" : "var qry = {'_queryFilter': org.forgerock.util.query.QueryFilter.equalTo('uid',
source.userName).toString()}; qry"
}
```

Common filter expressions

The ForgeRock REST API defines common filter expressions that enable you to form arbitrary queries using a number of supported filter operations. This query capability is the standard way to query data if no predefined query exists, and is supported for all managed and system objects.

Common filter expressions are useful in that they do not require knowledge of how the object is stored and do not require additions to the repository configuration.

Common filter expressions are called with the `_queryFilter` keyword. The following example uses a common filter expression to retrieve managed user objects whose user name is Smith:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
'http://localhost:8080/openidm/managed/user?_queryFilter=userName+eq+"smith"'
```

The filter is URL encoded in this example. The corresponding filter using the resource API would be:

```
openidm.query("managed/user", { "_queryFilter" : '/userName eq "smith"' });
```

Note that, this JavaScript invocation is internal and is not subject to the same URL-encoding requirements that a GET request would be. Also, because JavaScript supports the use of single quotes, it is not necessary to escape the double quotes in this example.

Parameterized queries

You can access managed objects in *JDBC repositories* using custom parameterized queries. Define these queries in your JDBC repository configuration, (`repo.*.json`), and call them by their `_queryId` .

Important

- Parameterized queries are not supported for system objects, or for DS repositories.
- All internal queries are filtered queries. Internal queries that reference a `queryId` are translated to filtered queries.

A typical query definition is as follows:

```
"query-all-ids" : "SELECT objectid FROM ${_dbSchema}.${_table} LIMIT ${int:_pageSize} OFFSET $
{int:_pagedResultsOffset}",
```

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

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

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
"http://localhost:8080/openidm/managed/user?_queryId=query-all-ids"
```

> **Note**
>
> In `repo.jdbc.json`, the `queries` configuration object has a property, `validInRelationshipQuery`, which is an array specifying the IDs of queries that use relationships. If you define parameterized queries that you expect to use as part of a relationship query, you must add the query ID to this array. If no query IDs are specified or if the property is absent, relationship information is not returned in query results, even if requested. For more information about relationships, refer to Relationships between objects.

Native query expressions

Native query expressions are supported for system objects only, and can be called directly.

You should only use native queries in situations where common query filters or parameterized queries are insufficient. For example, native queries are useful if the query needs to be generated dynamically.

The query expression is specific to the target resource and uses the native query language of that system resource.

Native queries are made using the `_queryExpression` keyword.

Construct queries

The `openidm.query` function lets you query managed and system objects. The query syntax is `openidm.query(id, params)`, where `id` specifies the object on which the query should be performed, and `params` provides the parameters that are passed to the query (the `_queryFilter`). For example:

```
var equalTo = org.forgerock.util.query.QueryFilter.equalTo;
queryParams = {
    "_queryFilter": equalTo("uid", value).toString()
};
openidm.query("managed/user", queryParams)
```

Over the REST interface, the query filter is specified as `_queryFilter=filter`, for example:

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

> **Note**
>
> In `_queryFilter` expressions, string values *must* use double-quotes. Numeric and boolean expressions should not use quotes.

When called over REST, you must URL encode the filter expression. The following examples show the filter expressions using the resource API and the REST API, but do not show the URL encoding, to make them easier to read.

For generic mappings, any fields that are included in the query filter (for example `userName` in the previous query), must be explicitly defined as *searchable*, if you have set the global `searchableDefault` to false. For more information, refer to Improving Generic Mapping Search Performance (JDBC).

The filter expression is constructed from the building blocks shown in this section. In these expressions the simplest json-pointer is a field of the JSON resource, such as `userName` or `id`. A JSON pointer can, however, point to nested elements.

> **Note**
>
> You can also use the negation operator (`!`) in query construction. For example, a `_queryFilter=!(userName+eq+"jdoe")` query would return every `userName` except for `jdoe`.

Comparison expressions

You can use comparison query filters for objects and object array properties that:

This is the associated JSON comparison expression: `json-pointer eq json-value`.

Example 1

```
"_queryFilter" : '/givenName eq "Dan"'
```

The following REST call returns the user name and given name of all managed users whose first name (`givenName`) is "Dan":

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/user?_queryFilter=givenName+eq+"Dan"&_fields=userName,givenName'
{
  "result": [
    {
      "givenName": "Dan",
      "userName": "dlangdon"
    },
    {
      "givenName": "Dan",
      "userName": "dcope"
    },
    {
      "givenName": "Dan",
      "userName": "dlanoway"
    }
  ],
  ...
}
```

Example 2

```
"_queryFilter" : "/stringArrayField eq 'foo'"
```

The following REST call returns role entries where a value within the `stringArrayField` array equals "foo":

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/role?_queryFilter=stringArrayField+eq+"foo"'
{
  "result": [
    {
      "_id": "admin2",
      "_rev": "0",
      "name": "admin2",
      "stringArrayField": [
        "foo",
        "bar"
      ]
    }
  ],
  ...
}
```

Additional information about PostgreSQL JSON functions.

This is the associated JSON comparison expression: `json-pointer co json-value`.

Example

```
"_queryFilter" : '/givenName co "Da"'
```

The following REST call returns the user name and given name of all managed users whose first name (`givenName`) contains "Da":

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/user?_queryFilter=givenName+co+"Da"&_fields=userName,givenName'
{
  "result": [
    {
      "givenName": "Dave",
      "userName": "djensen"
    },
    {
      "givenName": "David",
      "userName": "dakers"
    },
    {
      "givenName": "Dan",
      "userName": "dlangdon"
    },
    {
      "givenName": "Dan",
      "userName": "dcope"
    },
    {
      "givenName": "Dan",
      "userName": "dlanoway"
    },
    {
      "givenName": "Daniel",
      "userName": "dsmith"
    },
    ...
  ],
  "resultCount": 10,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
```

This is the associated JSON comparison expression: `json-pointer sw json-value`.

Example

```
 "_queryFilter" : '/sn sw "Jen"'
```

The following REST call returns the user names of all managed users whose last name (`sn`) starts with "Jen":

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/user?_queryFilter=sn+sw+"Jen"&_fields=userName'
{
  "result": [
    {
      "userName": "bjensen"
    },
    {
      "userName": "djensen"
    },
    {
      "userName": "cjenkins"
    },
    {
      "userName": "mjennings"
    }
  ],
  "resultCount": 4,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
```

This is the associated JSON comparison expression: `json-pointer lt json-value`.

Example

```
"_queryFilter" : '/employeeNumber lt 5000'
```

The following REST call returns the user names of all managed users whose `employeeNumber` is lower than 5000:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/user?
_queryFilter=employeeNumber+lt+5000&_fields=userName,employeeNumber'
{
  "result": [
    {
      "employeeNumber": 4907,
      "userName": "jnorris"
    },
    {
      "employeeNumber": 4905,
      "userName": "afrancis"
    },
    {
      "employeeNumber": 3095,
      "userName": "twhite"
    },
    {
      "employeeNumber": 3921,
      "userName": "abasson"
    },
    {
      "employeeNumber": 2892,
      "userName": "dcarter"
    },
    ...
  ],
  "resultCount": 4999,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
```

This is the associated JSON comparison expression: `json-pointer le json-value`.

Example

```
 "_queryFilter" : '/employeeNumber le 5000'
```

The following REST call returns the user names of all managed users whose `employeeNumber` is 5000 or less:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/user?
_queryFilter=employeeNumber+le+5000&_fields=userName,employeeNumber'
{
  "result": [
    {
      "employeeNumber": 4907,
      "userName": "jnorris"
    },
    {
      "employeeNumber": 4905,
      "userName": "afrancis"
    },
    {
      "employeeNumber": 3095,
      "userName": "twhite"
    },
    {
      "employeeNumber": 3921,
      "userName": "abasson"
    },
    {
      "employeeNumber": 2892,
      "userName": "dcarter"
    },
    ...
  ],
  "resultCount": 5000,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
```

This is the associated JSON comparison expression: `json-pointer gt json-value`

Example

```
 "_queryFilter" : '/employeeNumber gt 5000'
```

The following REST call returns the user names of all managed users whose `employeeNumber` is higher than 5000:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/user?
_queryFilter=employeeNumber+gt+5000&_fields=userName,employeeNumber'
{
  "result": [
    {
      "employeeNumber": 5003,
      "userName": "agilder"
    },
    {
      "employeeNumber": 5011,
      "userName": "bsmith"
    },
    {
      "employeeNumber": 5034,
      "userName": "bjensen"
    },
    {
      "employeeNumber": 5027,
      "userName": "cclarke"
    },
    {
      "employeeNumber": 5033,
      "userName": "scarter"
    },
    ...
  ],
  "resultCount": 1458,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
```

This is the associated JSON comparison expression: `json-pointer ge json-value`.

Example

```
 "_queryFilter" : '/employeeNumber ge 5000'
```

The following REST call returns the user names of all managed users whose `employeeNumber` is 5000 or greater:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/user?
_queryFilter=employeeNumber+ge+5000&_fields=userName,employeeNumber'
{
  "result": [
    {
      "employeeNumber": 5000,
      "userName": "agilder"
    },
    {
      "employeeNumber": 5011,
      "userName": "bsmith"
    },
    {
      "employeeNumber": 5034,
      "userName": "bjensen"
    },
    {
      "employeeNumber": 5027,
      "userName": "cclarke"
    },
    {
      "employeeNumber": 5033,
      "userName": "scarter"
    },
    ...
  ],
  "resultCount": 1457,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
```

Note

Although specific system endpoints also support `EndsWith` and `ContainsAllValues` queries, such queries are *not supported* for managed objects and have not been tested with all supported OpenICF connectors.

Presence expressions

The following examples show how you can build filters using a presence expression, shown as `pr`. The presence expression is a filter that returns all records with a given attribute.

A presence expression filter evaluates to `true` when a `json-pointer pr` matches any object in which the json-pointer is present, and contains a non-null value. Consider the following expression:

```
"_queryFilter" : '/mail pr'
```

The following REST call uses that expression to return the mail addresses for all managed users with a `mail` property:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/user?_queryFilter=mail+pr&_fields=mail'
{
  "result": [
    {
      "mail": "jdoe@exampleAD.com"
    },
    {
      "mail": "bjensen@example.com"
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
```

Depending on the connector, you can apply the presence filter on system objects. The following query returns the email address of all users in a CSV file who have the `email` attribute in their entries:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/system/csvfile/account?_queryFilter=email+pr&_fields=email'
{
  "result": [
    {
      "_id": "bjensen",
      "email": "bjensen@example.com"
    },
    {
      "_id": "scarter",
      "email": "scarter@example.com"
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": "MA%3D%3D",
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

> **Note**
>
> Not all connectors support the presence filter. In most cases, you can replicate the behavior of the presence filter with an "equals" (`eq`) query such as `_queryFilter=email+eq"*"`

Literal expressions

A literal expression is a boolean:

- `true` matches any object in the resource.
- `false` matches no object in the resource.

For example, you can list the `_id` of all managed objects as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/user?_queryFilter=true&_fields=_id'
{
  "result": [
    {
      "_id": "d2e29d5f-0d74-4d04-bcfe-b1daf508ad7c"
    },
    {
      "_id": "709fed03-897b-4ff0-8a59-6faaa34e3af6"
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
```

`In` expression clause

IDM provides limited support for the in expression clause. You can use this clause for queries on singleton string properties or arrays. The `in` query expression is not supported through the admin UI or for use by delegated administrators.

The `in` operator is shorthand for multiple `OR` conditions.

> **Note**
>
> The following example command includes escaped characters. For readability, the non-escaped URL syntax is:
>
> ```
> http://localhost:8080/openidm/managed/user?_pageSize=1000&_fields=userName&_queryFilter=/userName in
> '["user4a","user3a"]'
> ```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?
_pageSize=1000&_fields=userName&_queryFilter=userName%20in%20'%5B%22user4a%22%2C%22user3a%22%5D'"
{
  "result": [
    {
      "_id": "e32f9a3d-0039-4cb0-82d7-347cb808672e",
      "_rev": "000000000ae18357",
      "userName": "user3a"
    },
    {
      "_id": "120625c5-cfe7-48e7-b66a-6a0a0f9d2901",
      "_rev": "000000005ad98467",
      "userName": "user4a"
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

Filter expanded relationships

You can use `_queryFilter` to directly filter expanded relationships from a collection, such as `authzRoles`. The following example queries the `manager-int` authorization role of a user:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/b70293db-8743-45a7-9215-1ca8fd8a0073/authzRoles?
_queryFilter=name+eq+'manager-int'&_fields=*"
{
  "result": [
    {
      "_id": "b1d78144-7029-4135-8e73-85efe0a40b6b",
      "_rev": "00000000d4b8ab97",
      "_ref": "internal/role/c0a38233-c0f2-477d-8f18-f5485b7d002f",
      "_refResourceCollection": "internal/role",
      "_refResourceId": "c0a38233-c0f2-477d-8f18-f5485b7d002f",
      "_refProperties": {
        "_grantType": "",
        "_id": "b1d78144-7029-4135-8e73-85efe0a40b6b",
        "_rev": "00000000d4b8ab97"
      },
      "name": "manager-int",
      "description": "manager-int-desc",
      "temporalConstraints": null,
      "condition": null,
      "privileges": null
    }
  ],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

Warning

You can use `_queryFilter` on fields within `_refProperties` when using DS as your repository. This functionality is not available if you are using a JDBC repository.

Complex expressions

You can combine expressions using the boolean operators `and`, `or`, and `!` (not). The following example queries managed user objects located in London, with last name Jensen:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/user/?
_queryFilter=city+eq+"London"and+sn+eq"Jensen"&_fields=userName,givenName,sn'
{
  "result": [
    {
      "sn": "Jensen",
      "givenName": "Clive",
      "userName": "cjensen"
    },
    {
      "sn": "Jensen",
      "givenName": "Dave",
      "userName": "djensen"
    },
    {
      "sn": "Jensen",
      "givenName": "Margaret",
      "userName": "mjensen"
    }
  ],
  "resultCount": 3,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
```

Filter objects in arrays

Use query grouping to perform your query on properties within an array. For example, to query `effectiveRoles` for users who have the `testManagedRole`, check the `_refResourceId` inside the `effectiveRoles` array:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/managed/user/?_queryFilter=/effectiveRoles\[/
_refResourceId+eq+"testManagedRole"]&_fields=userName,givenName,sn,effectiveRoles'
{
  "result": [
    {
      "_id": "917bc052-ef39-4add-ae05-0a278e2de9c0",
      "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1565",
      "userName": "scarter",
      "sn": "Carter",
      "givenName": "Steven",
      "effectiveRoles": [
        {
          "_refResourceCollection": "managed/role",
          "_refResourceId": "testManagedRole",
          "_ref": "managed/role/testManagedRole"
        }
      ]
    },
    {
      "_id": "aca0042c-9f4c-4ad5-8cf7-aca0adeb3470",
      "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1545",
      "userName": "jdoe",
      "sn": "Doe",
      "givenName": "John",
      "effectiveRoles": [
        {
          "_refResourceCollection": "managed/role",
          "_refResourceId": "testManagedRole",
          "_ref": "managed/role/testManagedRole"
        }
      ]
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

Note

Because `curl` uses brackets (`[]` , `{}`) for processing, you need to escape your brackets with a `\` . This may be unnecessary in cases where you are using a different method to call IDM.

> **Important**
>
> This syntax is only available when using DS or PostgreSQL as your repository.
> When using a PostgreSQL repository and querying an array, properties that are a string, boolean, number, or object are supported. However, arrays are not supported (you can't filter on an array within an array).

Page query results

The common filter query mechanism supports paged query results for managed objects, and for some system objects, depending on the system resource. There are two ways to page objects in a query:

- Using a cookie based on the value of a specified sort key.
- Using an offset that specifies how many records should be skipped before the first result is returned.

These methods are implemented with the following query parameters:

`_pagedResultsCookie`

Opaque cookie used by the server to keep track of the position in the search results. The format of the cookie is a base-64 encoded version of the value of the unique sort key property. The value of the returned cookie is URL-encoded to prevent values such as `+` from being incorrectly translated.

You cannot page results without sorting them (using the `_sortKeys` parameter). If you do not specify a sort key, the `_id` of the record is used as the default sort key. At least one of the specified sort key properties must be a unique value property, such as `_id`.

> **Tip**
>
> For paged searches on generic mappings, you should sort on the `_id` property, because this is the only property that is stored outside of the JSON blob. If you sort on something other than `_id`, the search will incur a performance hit because IDM effectively has to pull the entire result set, and then sort it.

The server provides the cookie value on the first request. You should then supply the cookie value in subsequent requests until the server returns a null cookie, meaning that the final page of results has been returned.

The `_pagedResultsCookie` parameter is supported only for filtered queries, that is, when used with the `_queryFilter` parameter. You cannot use the `_pagedResultsCookie` with a `_queryId`.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and cannot be used together.

Paged results are enabled only if the `_pageSize` is a non-zero integer.

`_pagedResultsOffset`

Specifies the index within the result set of the number of records to be skipped before the first result is returned. The format of the `_pagedResultsOffset` is an integer value. When the value of `_pagedResultsOffset` is greater than or equal to 1, the server returns pages, starting after the specified index.

This request assumes that the `_pageSize` is set, and not equal to zero.

For example, if the result set includes 10 records, the `_pageSize` is 2, and the `_pagedResultsOffset` is 6, the server skips the first 6 records, then returns 2 records, 7 and 8. The `_remainingPagedResults` value would be 2, the last two records (9 and 10) that have not yet been returned.

If the offset points to a page beyond the last of the search results, the result set returned is empty.

`_pageSize`

An optional parameter indicating that query results should be returned in pages of the specified size. For all paged result requests other than the initial request, a cookie should be provided with the query request.

The default behavior is not to return paged query results. If set, this parameter should be an integer value, greater than zero.

When a `_pageSize` is specified, and non-zero, the server calculates the `totalPagedResults`, in accordance with the `totalPagedResultsPolicy`, and provides the value as part of the response. If a count policy is specified (`_totalPagedResultsPolicy=EXACT`, The `totalPagedResults` returns the total result count. If no count policy is specified in the query, or if `_totalPagedResultsPolicy=NONE`, result counting is disabled, and the server returns a value of -1 for `totalPagedResults`. The following example shows a query that requests two results with a `totalPagedResultsPolicy` of `EXACT`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?
_queryFilter=true&_pageSize=2&_totalPagedResultsPolicy=EXACT"
{
  "result": [
    {
      "_id": "adonnelly",
      "_rev": "0",
      "userName": "adonnelly",
      "givenName": "Abigail",
      "sn": "Donnelly",
      "telephoneNumber": "12345678",
      "active": "true",
      "mail": "adonnelly@example.com",
      "accountStatus": "active",
      "effectiveRoles": [],
      "effectiveAssignments": []
    },
    {
      "_id": "bjensen",
      "_rev": "0",
      "userName": "bjensen",
      "givenName": "Babs",
      "sn": "Jensen",
      "telephoneNumber": "12345678",
      "active": "true",
      "mail": "bjensen@example.com",
      "accountStatus": "active",
      "effectiveRoles": [],
      "effectiveAssignments": []
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": "eyIvX2lkIjoiYm11cnJheSJ9",
  "totalPagedResultsPolicy": "EXACT",
  "totalPagedResults": 22,
  "remainingPagedResults": -1
}
```

The `totalPagedResults` and `_remainingPagedResults` parameters are not supported for all queries. Where they are not supported, their returned value is always `-1`. In addition, counting query results using these parameters is not currently supported for a ForgeRock Directory Services (DS) repository.

Requesting the total result count (with `_totalPagedResultsPolicy=EXACT`) incurs a performance cost on the query.

Queries that return large data sets will have a significant impact on heap requirements, particularly if they are run in parallel with other large data requests. To avoid out of memory errors, analyze your data requirements, set the heap configuration appropriately, and modify access controls to restrict requests on large data sets.

Sort query results

For common filter query expressions, you can sort the results of a query using the `_sortKeys` parameter. This parameter takes a comma-separated list as a value and orders the way in which the JSON result is returned, based on this list.

The `_sortKeys` parameter is not supported for predefined queries.

Note

When using DS as a repo:

- Pagination using `_pageSize` is recommended if you intend to use `_sortKeys`. If you do not paginate your query, the data you are querying must be indexed in DS.
- When viewing data that is persisted in DS and sorted by un-indexed `_sortKeys`, the `_pageSize` parameter must be less than or equal to the `index-entry-limit` as configured in DS (default value is 4000).

For more information about how to set up indexes in DS, refer to Indexes in the *DS Configuration Guide*.

The following query returns all users with the `givenName` Dan, and sorts the results alphabetically, according to surname (`sn`):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/system/ldap/account?
_queryFilter=givenName+eq+"Dan"&_fields=givenName,sn&_sortKeys=sn'
{
  "result": [
    {
      "sn": "Cope",
      "givenName": "Dan"
    },
    {
      "sn": "Langdon",
      "givenName": "Dan"
    },
    {
      "sn": "Lanoway",
      "givenName": "Dan"
    }
  ],
  "resultCount": 3,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
```

Note

When you query a relationship field, fields that belong to the related object are not available as `_sortKeys`. For example, if you query a list of a manager's reports, you cannot sort by the reports' last names. This is because the available `_sortKeys` are based on the object being queried, which, in the case of relationships, is actually a list of references to other objects, not the objects themselves.

Recalculate virtual property values in queries

For managed objects IDM includes an `onRetrieve` script hook that enables you to recalculate property values when an object is retrieved as the result of a query. To use the `onRetrieve` trigger, the query must include the `executeOnRetrieve` parameter, for example:

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

If a query includes `executeOnRetrieve`, the query recalculates virtual property values, based on the current state of the system. The result of the query will be the same as a `read` on a specific object, because reads always recalculate virtual property values.

If a query does not include `executeOnRetrieve`, the query returns the virtual properties of an object, based on the value that is persisted in the repository. Virtual property values are not recalculated.

For performance reasons, `executeOnRetrieve` is `false` by default.

> **Note**
>
> Virtual properties that use `queryConfig` for calculation instead of an `onRetrieve` script are not recalculated by `executeOnRetrieve`. These properties are recalculated only when there is a change (such as adding or removing a role affecting `effectiveRoles`, or a temporal constraint being triggered or changed).

Upload files to the server

IDM provides a generic file upload service that lets you upload and save files either to the filesystem or to the repository. The service uses the `multipart/form-data` Content-Type to accept file content, store it, and return that content when it is called over the REST interface.

To configure the file upload service, add one or more `file-description.json` files to your project's `conf` directory, where description provides an indication of the purpose of the upload service. For example, you might create a `file-images.json` configuration file to handle uploading image files. Each file upload configuration file sets up a separate instance of the upload service. The description in the filename also specifies the endpoint at which the file service will be accessible over REST. In the previous example, `file-images.json`, the service would be accessible at the endpoint `openidm/file/images`.

A sample file upload service configuration file is available in the `/path/to/openidm/samples/example-configurations/conf` directory. The configuration is as follows:

```
{
    "enabled" : true,
    "fileHandler" : {
        "type" : file handler type,
        "root" : directory
    }
}
```

The service supports two *file handlers*— `file` and `repo`. The file handlers are configured as follows:

- `"type" : "file"` specifies that the uploaded content will be stored in the filesystem. If you use the `file` type, you must specify a `root` property to indicate the directory (relative to the IDM installation directory) in which uploaded content is stored. In the following example, uploaded content is stored in the `/path/to/openidm/images` directory:

```
{
    "enabled" : true,
    "fileHandler" : {
        "type" : "file",
        "root" : "images"
    }
}
```

  You cannot use the file upload service to access any files outside the configured `root` directory.

  > **Warning**
  >
  > If `root` is configured to be an empty string, do not grant access to the file upload service to end users. When `type` is configured as `file`, ensure that `root` is configured to be a directory.

- `"type" : "repo"` specifies that the uploaded content will be stored in the repository. The `root` property does not apply to the repository file handler so the configuration is as follows:

```
{
    "enabled" : true,
    "fileHandler" : {
        "type" : "repo"
    }
}
```

The file upload service performs a multi-part CREATE operation. Each upload request includes two `--form` options. The first option indicates that the uploaded file content will be converted to a base 64-encoded string and inserted into the JSON object as a field named `content` with the following structure:

```
{
    "content" : {
        "$ref" : "cid:filename#content"
    }
}
```

The second `--form` option specifies the file to be uploaded, and the file type. The request loads the entire file into memory, so file size will be constrained by available memory.

You can upload any mime type using this service; however, you must specify a safelist of mime types that can be *retrieved* over REST. If you specify a mime type that is not in the safelist during retrieval of the file, the response content defaults to `application/json`. To configure the list of supported mime types, specify a comma-separated list as the value of the `org.forgerock.json.resource.http.safemimetypes` property in the `conf/system.properties` file. For example:

```
org.forgerock.json.resource.http.safemimetypes=application/json,application/pkix-cert,application/x-pem-file
```

You can only select from the following list:

- `image/*`
- `text/plain`
- `text/css`
- `text/csv`
- `application/json`
- `application/pkix-cert`
- `application/x-pem-file`

The following request uploads an image (PNG) file named `test.png` to the filesystem. The file handler configuration file provides the REST endpoint. In this case `openidm/file/images` references the configuration in the `file-images.json` file:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--form 'json={"content" : {"$ref" : "cid:test#content"}};type=application/json' \
--form 'test=@test.png;type=image/png' \
--request PUT \
"http://localhost:8080/openidm/file/images/test.png"
{
  "_id": "test.png",
  "content": "aW1hZ2UvcG5n"
}
```

Note that the resource ID is derived directly from the upload filename — system-generated IDs are not supported.

The following request uploads a stylesheet (`css`) file named `test.css` to the same location on the filesystem as the previous request:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--form 'json={"content" : {"$ref" : "cid:test#content"}};type=application/json' \
--form '@test.css;type=text/css' \
--request PUT \
"http://localhost:8080/openidm/file/images/test.css"
{
  "_id": "test.css",
  "content": "aW1hZ2UvY3N2"
}
```

Files uploaded to the repository are stored as JSON objects in the `openidm.files` table. The following request uploads the same image (PNG) file (`test.png`) to the repository:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--form 'json={"content" : {"$ref" : "cid:test#content"}};type=application/json' \
--form 'test=@test.png;type=image/png' \
--request PUT \
"http://localhost:8080/openidm/file/repo/test.png"
{
  "_id": "test.png",
  "_rev": "00000000970b4454",
  "content": "aW1hZ2UvcG5n"
}
```

Note that the preceding example assumes the following file upload service configuration (in `file-repo.json` :

```
 {
     "enabled" : true,
     "fileHandler" : {
         "type" : "repo"
     }
 }
```

The file type is not stored with the file. By default, a READ on uploaded file content returns the content as a base 64-encoded string within the JSON object. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/file/repo/test.png"
{
  "_id": "test.png",
  "_rev": "00000000970b4454",
  "content": "aW1hZ2UvcG5n"
}
```

Your client can retrieve the file in the correct format by specifying the `content` and `mimeType` parameters in the read request. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/file/repo/test.css?_fields=content&_mimeType=text/css"
```

To delete uploaded content, send a DELETE request as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/file/repo/test.png"
{
  "_id": "test.png",
  "_rev": "00000000970b4454",
  "content": "aW1hZ2UvcG5n"
}
```

Data models and objects reference

You can customize a variety of objects that can be addressed via a URL or URI. IDM can perform a common set of functions on these objects, such as CRUDPAQ (create, read, update, delete, patch, action, and query).

Depending on how you intend to use them, different object types are appropriate.

Object Types

Object Type	Intended Use	Special Functionality
Managed objects	Serve as targets and sources for synchronization, and to build virtual identities.	Provide appropriate auditing, script hooks, declarative mappings and so forth in addition to the REST interface.
Configuration objects	Ideal for look-up tables or other custom configuration, which can be configured externally like any other system configuration.	Adds file view, REST interface, and so forth
Repository objects	The equivalent of arbitrary database table access. Appropriate for managing data purely through the underlying data store or repository API.	Persistence and API access

Object Type	Intended Use	Special Functionality
System objects	Representation of target resource objects, such as accounts, but also resource objects such as groups.	
Audit objects	Houses audit data in the repository.	
Links	Defines a relation between two objects.	

Managed objects reference

A *managed object* is an object that represents the identity-related data managed by IDM. Managed objects are stored in the IDM repository. All managed objects are JSON-based data structures.

Managed object schema

IDM provides a default schema for typical managed object types, such as users and roles, but does not control the structure of objects that you store in the repository. You can modify or extend the schema for the default object types, and you can set up a new managed object type for any item that can be collected in a data set.

The `_rev` property of a managed object is reserved for internal use, and is not explicitly part of its schema. This property specifies the revision of the object in the repository. 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 validation is performed by the policy service and can be configured according to the requirements of your deployment.

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. Such properties are named *virtual properties*. The value of a virtual property is computed only when that property is retrieved.

Data consistency

Single-object operations are consistent within the scope of the operation performed, limited by the capabilities of the underlying data store. Bulk operations have no consistency guarantees. IDM does not expose any transactional semantics in the managed object access API.

For information on conditional header access through the REST API, refer to Conditional Operations.

Managed object triggers

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

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.

postCreate

Called after the creation of a new object is complete.

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 `oldObject` and `newObject` variables are in-scope for the trigger. The `oldObject` represents a complete object, as retrieved from the data store. The trigger can elect to change `newObject` properties. If, as a result of the trigger, the values of the `oldObject` and `newObject` are identical (that is, update is reverted), the update ends prematurely, but successfully. Throwing an exception causes the update to fail.

postUpdate

Called after an update request is complete.

onDelete

Called upon a request to delete an object. Throwing an exception causes the deletion to fail.

postDelete

Called after an object is deleted.

onSync

Called when a managed object is changed, and the change triggers an implicit synchronization operation. The implicit synchronization operation is triggered by calling the sync service, which attempts to go through all the configured managed-system mappings. The sync service returns either a response or an error. For both the response and the error, the script that is referenced by the `onSync` hook is called.

You can use this hook to inject business logic when the sync service either fails or succeeds to synchronize all applicable mappings. For an example of how the `onSync` hook is used to revert partial successful synchronization operations, refer to Synchronization Failure Compensation.

Object storage triggers

An object-scoped trigger applies to an entire object. Unless otherwise specified, the object itself is in scope for the trigger.

onValidate

Validates an object prior to its storage in the data store. If an exception is thrown, the validation fails and the object is not stored.

onStore

Called just prior to when an object is stored in the data store. Typically used to transform an object just prior to its storage (for example, encryption).

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 in the data store. If an exception is thrown, the validation fails and the property is not stored.

onRetrieve

Called on all requests that return a single object: read, create, update, patch, and delete.

`onRetrieve` is called on queries only if `executeOnRetrieve` is set to `true` in the query request parameters. If `executeOnRetrieve` is not passed, or if it is `false`, the query returns previously persisted values of the requested fields. This behavior avoids performance problems when executing the script on all results of a query.

onStore

Called before an object is stored in the data store. Typically used to transform a given property before its object is stored.

Storage trigger sequences

Triggers are executed in the following order:

Object Retrieval Sequence

1. Retrieve the raw object from the data store
2. The `executeOnRetrieve` boolean is used to check whether property values should be recalculated. The sequence continues if the boolean is set to `true`.
3. Call object `onRetrieve` trigger
4. Per-property within the object, call property `onRetrieve` trigger

Object Storage Sequence

1. Per-property within the object:
    - Call property `onValidate` trigger
    - Call object `onValidate` trigger

2. Per-property trigger within the object:
    - Call property `onStore` trigger
    - Call object `onStore` trigger
    - Store the object with any resulting changes to the data store

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.

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,
  "actions"   : script object,
  "onCreate"  : script object,
  "onDelete"  : script object,
  "onRead"    : script object,
  "onRetrieve": script object,
  "onStore"   : script object,
  "onSync"    : script object,
  "onUpdate"  : script object,
  "onValidate": script object,
  "postCreate": script object,
  "postDelete": script object,
  "postUpdate": script object,
  "schema"    : {
    "id"        : urn,
    "icon"      : string,
    "mat-icon"  : string,
    "order"     : [ list of properties],
    "properties": { property-configuration objects },
    "$schema"   : "http://json-schema.org/draft-03/schema",
    "title"     : "User",
    "viewable"  : true
  }
}
```

name

string, required

The name of the managed object. Used to identify the managed object in URIs and identifiers.

actions

script object, optional

A custom script that initiates an action on the managed object. For more information, refer to Register custom scripted actions.

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 can change the object. If an exception is thrown, the create 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.

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 can change the object. If an exception is thrown, the read aborts with an exception.

onRetrieve

script object, optional

A script object to trigger when an object is retrieved from the repository. The object that was retrieved is provided in the root scope as an `object` property. The script can 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 can change the object. If an exception is thrown, then object storage fails.

onSync

script object, optional

A script object to trigger when a change to a managed object triggers an implicit synchronization operation. The script has access to the `syncResults` object, the `request` object, the state of the object before the change (`oldObject`) and the state of the object after the change (`newObject`). The script can change the object.

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 can change the `newObject` . If an exception is thrown, the update 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.

postCreate

script object, optional

A script object to trigger after an object is created, but before any targets are synchronized.

postDelete

script object, optional

A script object to trigger after a delete of an object is complete, but before any further synchronization. The value of the deleted object is provided in the root scope as an `oldObject` property.

postUpdate

script object, optional

A script object to trigger after an update to an object is complete, but before any targets are synchronized. The value of the object before the update is provided in the root scope as an `oldObject` property. The value of the object after the update is provided in the root scope as a `newObject` property.

schema

json-schema object, optional

The schema to use to validate the structure and content of the managed object, and how the object is displayed in the UI. The schema-object format is defined by the JSON Schema specification.

The `schema` property includes the following additional elements:

icon

string, optional

The name of the Font Awesome icon to display for this object in the UI. Only applies to standalone IDM.

mat-icon

string, optional

The name of the Material Design Icon to display for this object in the UI. Only applies to IDM as part of the ForgeRock Identity Platform.

id

urn, optional

The URN of the managed object, for example, `urn:jsonschema:org:forgerock:openidm:managed:api:Role`.

order

list of properties, optional

The order in which properties of this managed object are displayed in the UI.

properties

list of property configuration objects, optional

A list of property specifications. For more information, refer to Property Configuration Properties.

$schema

url, optional

Link to the JSON schema specification.

title

string, optional

The title of this managed object in the UI.

viewable

boolean, optional

Whether this object is visible in the UI.

Property configuration properties

Each managed object property, identified by its `property-name`, can have the following configurable properties:

```
"property-name" : {
  "comparison"      : string,
  "description"     : string,
  "encryption"      : property-encryption object,
  "isPersonal"      : boolean true/false,
  "isProtected"     : boolean true/false,
  "isVirtual"       : boolean true/false,
  "items"           : {
     "id"                  : urn,
     "properties"          : property-config object,
     "resourceCollection"  : property-config object,
     "reversePropertyName" : string,
     "reverseRelationship" : boolean true/false,
     "title"               : string,
     "type"                : string,
     "validate"            : boolean true/false,
  },
  "onRetrieve"      : script object,
  "onStore"         : script object,
  "onValidate"      : script object,
  "pattern"         : string,
  "policies"        : policy object,
  "required"        : boolean true/false,
  "returnByDefault" : boolean true/false,
  "scope"           : string,
  "searchable"      : boolean true/false,
  "secureHash"      : property-hash object,
  "title"           : string,
  "type"            : data type,
  "usageDescription": string,
  "userEditable"    : boolean true/false,
  "viewable"        : boolean true/false,
}
```

comparison

string, optional

Specifies whether to use array ordered or unordered comparison for synchronization. The value can be either `"ordered"` or `"unordered"`. The `"ordered"` value indicates that the array order matters with regard to detecting changes for sync. The `"unordered"` value indicates that the array order does not matter with regard to detecting changes for sync.

Relationship and virtual property array fields default to unordered comparisons. All other fields default to ordered comparisons.

If you're using explicit mappings with a DS repository, you can't use ordered array comparisons.

description

string, optional

A brief description of the property.

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.

isPersonal

boolean, true/false

Designed to highlight personally identifying information. By default, `isPersonal` is set to `true` for `userName` and `postalAddress`.

isProtected

boolean, true/false

Specifies whether reauthentication is required if the value of this property changes.

isVirtual

boolean, true/false

Specifies whether the property takes a static value, or whether its value is calculated dynamically as the result of a script.

The most recently calculated value of a virtual property is persisted by default. The persistence of virtual property values allows IDM to compare the new value of the property against the last calculated value, and therefore to detect change events during synchronization.

Virtual property values are not persisted by default if you're using an explicit mapping.

items

property-configuration object, optional

For `array` type properties, defines the elements in the array. `items` can include the following sub-properties:

id

urn, optional

The URN of the property, for example, `urn:jsonschema:org:forgerock:openidm:managed:api:Role:members:items`.

properties

property configuration object, optional

A list of properties, and their configuration, that make up this items array. For example, for a relationship type property:

```
"properties" : {
    "_ref" : {
        "description" : "References a relationship from a managed object",
        "type" : "string"
    },
    "_refProperties" : {
        "description" : "Supports metadata within the relationship",
        ...
    }
}
```

resourceCollection

property configuration object, optional

The collection of resources (objects) on which this relationship is based (for example, `managed/user` objects).

reversePropertyName

string, optional

For `relationship` type properties, specifies the corresponding property name in the case of a reverse relationship. For example, a `roles` property might have a `reversePropertyName` of `members`.

reverseRelationship

boolean, true or false.

For `relationship` type properties, specifies whether the relationship exists in both directions.

title

string, optional

The title of array items, as displayed in the UI, for example `Role Members Items`.

type

string, optional

The array type, for example `relationship`.

validate

boolean, true/false

For reverse relationships, specifies whether the relationship should be validated.

onRetrieve

script object, optional

A script object to trigger once a property is retrieved from the repository. That property may be one of two related variables: `property` and `propertyName`. The property that was retrieved is provided in the root scope as the `propertyName` variable; its value is provided as the `property` variable. 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. That property may be one of two related variables: `property` and `propertyName`. The property that was retrieved is provided in the root scope as the `propertyName` variable; its value is provided as the `property` variable. If an exception is thrown, then object storage fails.

onValidate

script object, optional

A script object to trigger when the property requires validation. The value of the property to be validated is provided in the root scope as the `property` property. If an exception is thrown, validation fails.

pattern

string, optional

Any specific pattern to which the value of the property must adhere. For example, a property whose value is a date might require a specific date format. Patterns specified here must follow regular expression syntax.

policies

policy object, optional

Any policy validation that must be applied to the property.

required

boolean, true/false

Specifies whether the property must be supplied when an object of this type is created.

returnByDefault

boolean, true/false

For virtual properties, specifies whether the property will be returned in the results of a query on an object of this type if it is not explicitly requested. Virtual attributes are not returned by default.

scope

string, optional

Specifies whether the property should be filtered from HTTP/external calls. The value can be either `"public"` or `"private"`. `"private"` indicates that the property should be filtered, `"public"` indicates no filtering. If no value is set, the property is assumed to be public and thus not filtered.

searchable

boolean, true/false

Specifies whether this property can be used in a search query on the managed object. A searchable property is visible in the End User UI. False by default.

secureHash

property-hash object, optional

Specifies the configuration for hashing of the property value in the repository. If omitted or null, the property is not hashed.

title

string, required

A human-readable string, used to display the property in the UI.

type

data type, required

The data type for the property value; can be String, Array, Boolean, Number, Object, or Resource Collection.

usageDescription

string, optional

Designed to help end users understand the sensitivity of a property such as a telephone number.

userEditable

boolean, true/false

Specifies whether users can edit the property value in the UI. This property applies in the context of the End User UI, in which users are able to edit certain properties of their own accounts. False by default.

viewable

boolean, true/false

Specifies whether this property is viewable in the object's profile in the UI. True by default.

Script Object Properties

```
{
  "type"   : "text/javascript",
  "source": string
}
```

type

string, required

IDM supports `"text/javascript"` and `"groovy"`.

source, file

string, required (only one, source or file is required)

Specifies the source code of the script to be executed (if the keyword is "source"), or a pointer to the file that contains the script (if the keyword is "file").

Property 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 IDM cryptography service keystore used to encrypt the property.

Property Hash Object

```
{
    "algorithm" : string,
    "type" : string
}
```

algorithm

string, required

The algorithm that should be used to hash the value.

For a list of supported hash algorithms, refer to Salted Hash Algorithms.

type

string, optional

The type of hashing. Currently only salted hash is supported. If this property is omitted or null, the default `"salted-hash"` is used.

Custom managed objects

Managed objects are inherently fully user definable and customizable. Like all 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 *managed objects* comes from the intention that IDM stores and manages these objects, as opposed to system objects that are present in external systems.

IDM 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 as 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.

Set up a managed object type

To set up a managed object, you define the object in your project's managed object configuration. This simple example adds a `foobar` object declaration after the `user` object type:

```
{
    "objects": [
        {
            "name": "user"
        },
        {
            "name": "foobar"
        }
    ]
}
```

Manipulate 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. For more information, refer to Resource mapping.

Manipulate 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 IDM 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. For more information, refer to Scripting.

IDM supports concurrency through a multi version concurrency control (MVCC) mechanism. Each time an object changes, IDM 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.

Create objects

The following script example creates an object type.

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

Update 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, IDM rejects the update, and you must either retry or inspect the concurrent modification.

Patch objects

You can partially update a managed or system object using the patch method, which changes only the specified properties of the object.

The following script example updates an object type.

```
openidm.patch("managed/foobar/myidentifier", rev, value)
```

The patch method supports a revision of `"null"`, which effectively disables the MVCC mechanism, that is, changes are applied, regardless of revision. In the REST interface, this matches the `If-Match: "*"` condition supported by patch. Alternatively, you can omit the "If-Match: *" header.

For managed objects, the API supports patch by query, so the caller does not need to know the identifier of the object to change.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '[
  {
    "operation": "replace",
    "field": "/password",
    "value": "Passw0rd"
  }
]' \
"http://localhost:8080/openidm/managed/user?_action=patch&_queryFilter=userName+eq+'DDOE'"
```

Delete 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, IDM rejects deletion, and you must either retry or inspect the concurrent modification.

Read objects

The following script example reads an object type.

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

Query object sets

You can query managed objects using common query filter syntax. The following script example queries managed user objects whose `userName` is smith.

```
var qry = {
    "_queryFilter" : "/userName eq \"smith\""
};
val = openidm.query("managed/user", qry);
```

For more information, refer to Define and call data queries.

Access managed objects through the REST API

IDM 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. For more information, refer to the REST API reference.

IDM requires authentication to access the REST API. The authentication configuration is specified in your project's `conf/authentication.json` file. The default authorization filter script is `openidm/bin/defaults/script/router-authz.js`. For more information, refer to Authorization and roles.

Configuration objects

IDM provides an extensible configuration to allow you to leverage regular configuration mechanisms.

Unlike native the IDM configuration, which is interpreted automatically and can start new services, IDM stores custom configuration objects and makes them available to your code through the API.

For an introduction to the standard configuration objects, refer to Server configuration.

When To use custom configuration objects

Configuration objects are ideal for metadata and settings that need not be included in the data to reconcile. Use configuration objects for data that does not require audit logs, and does not serve directly as a target or source for mappings.

Although you can set and manipulate configuration objects programmatically and manually, configuration objects are expected to change slowly, through both manual file updates and 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 might be more appropriate.

Configuration objects file and REST payload formats

By default, IDM maps configuration objects to JSON representations.

IDM represents objects internally in plain, native types like maps, lists, strings, numbers, booleans, null. The object model is restricted to simple types so that mapping objects to external representations is easy.

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"
}
```

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` .

IDM supports REST mappings for create, read, update, delete, patch, and query of configuration objects.

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, query, and delete operations.

IDM supports concurrency through a multi version concurrency control mechanism. Each time an object changes, IDM 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.

Creating objects

The following script example creates an object type.

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

Updating objects

The following script example updates a custom configuration object type.

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

Deleting objects

The following script example deletes a custom configuration object type.

```
openidm.delete("config/custom/myconfig")
```

Reading objects

The following script example reads an object type.

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

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.

Audit objects

Audit objects contain audit data selected for local storage in repository.

Links

Link objects define relations between source objects and target objects, usually relations between managed objects and system objects. The link relationship is established by provisioning activity that either results in a new account on a target system, or a reconciliation or synchronization scenario that takes a `LINK` action.

Authentication and authorization

This guide covers authentication, authorization, and delegated administration.

Authentication

Authenticate users securely.

Authorization & roles

The authorization configuration grants rights to users based on their roles.

Delegated administration

Use privileges to give fine-grained administrative access to specific users.

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

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

Authentication

Authentication is the process of verifying who is requesting access to a resource. The user or application making the request presents credentials, making it possible to prove that the requester is who they claim to be. The goal is to authorize access to specific IDM resources, depending on the confirmed identity of the user or application making the request.

IDM supports two authentication modes:

- Using one or more of the *classic* IDM authentication modules.
- Configuring IDM as an OAuth2 Resource Server in a *platform deployment* using AM as the Identity Provider.

IDM and HTTP basic authentication

HTTP basic authentication is a simple challenge and response mechanism whereby the client submits a user ID and password to the server. IDM understands the authorization header of the HTTP basic authentication contract. However, it deliberately does not use the full HTTP basic authentication contract and does not cause the browser built-in mechanism to prompt for username and password. It also understands utilities, such as `curl` and Postman, that can send the username and password in the Authorization header.

In general, the HTTP basic authentication mechanism does not work well with client side web applications, and applications that need to render their own login screens. Because the browser stores and sends the username and password with each request, HTTP basic authentication has significant security vulnerabilities. You can therefore send the username and password via the authorization header, and IDM returns a token for subsequent access.

Access to the IDM REST interface *requires* that the client authenticate. User self-registration requires anonymous access. For this purpose, IDM includes an `anonymous` user, with the password `anonymous`. For more information, refer to Internal Users.

The examples in this documentation use the IDM authentication headers in all REST examples, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
...
```

Password changes

Changing passwords can expose a server to potential security risks. An insecure password reset process can allow attackers to reset the passwords of other users in order to bypass authentication and gain access to user accounts.

Re-authentication forces users or clients to confirm their identity even this identity was verified previously. When passwords are changed over REST, using a PUT or PATCH request, IDM requires the `X-OpenIDM-Reauth-Password` header. If this header is absent, the server returns a `403` error.

For example, the following password change request fails:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "If-Match: *" \
--request PUT \
--data '{
  "userName": "bjensen",
  "givenName": "Babs",
  "sn": "Jensen",
  "mail": "babs.jensen@example.com",
  "telephoneNumber": "555-123-1234",
  "password": "NewPassw0rd"
}' \
"https://localhost:8443/openidm/managed/user/0638da14-e02e-4904-9076-b8ce8f700eb4"
{
  "code": 403,
  "reason": "Forbidden",
  "message": "Access denied"
}
```

The same request, including the `X-OpenIDM-Reauth-Password` header, succeeds:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "X-OpenIDM-Reauth-Password: Passw0rd" \
--header "If-Match: *" \
--request PUT \
--data '{
  "userName": "bjensen",
  "givenName": "Babs",
  "sn": "Jensen",
  "mail": "babs.jensen@example.com",
  "telephoneNumber": "555-123-1234",
  "password": "NewPassw0rd"
}' \
"https://localhost:8443/openidm/managed/user/0638da14-e02e-4904-9076-b8ce8f700eb4"
{
  "_id": "0638da14-e02e-4904-9076-b8ce8f700eb4",
  "_rev": "00000000fa190282",
  "userName": "bjensen",
  "givenName": "Babs",
  "sn": "Jensen",
  "mail": "babs.jensen@example.com",
  "telephoneNumber": "555-123-1234",
  ...
}
```

Character encoding in authentication headers

You can use encoded characters in all three IDM authentication headers (`X-OpenIDM-Username` , `X-OpenIDM-Password` , and `X-OpenIDM-Reauth-Password`). This lets you use non-ASCII characters in these header values. The RFC 5987-encoding is automatically detected and decoded when present. The following character sets are supported:

- UTF-8
- ISO 8859-1

The following command shows a request for a user (openidm-admin) whose password is `Passw£rd123` . The Unicode `£` sign (U+00A3) is encoded into the octet sequence C2 A3 using UTF-8 character encoding, then percent-encoded:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: UTF-8''Passw%C2%A3rd123" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/managed/user?_queryFilter=true&_fields=_id"
```

Authenticate users

IDM stores two types of users in its repository—internal users and managed users.

Internal users

Internal users are special user accounts that are stored separately from regular users to protect them from any reconciliation or synchronization processes. When IDM first starts up, it creates three internal users in the repository by default— `openidm-admin`, `anonymous`, and `idm-provisioning`:

openidm-admin

This user serves as the top-level administrator and has full access to all IDM resources. This account provides a fallback mechanism in the event that other users are locked out of their accounts. Do not use `openidm-admin` for regular tasks. Under normal circumstances, the `openidm-admin` account does not represent a regular user, so audit log records for this account do not represent the actions of any real person.

The default password for the `openidm-admin` user is `openidm-admin`. In production environments, you should change this password, as described in Change the Administrator User Password. The new password is symmetrically encrypted as it is changed.

anonymous

This user enables anonymous access to IDM. It is used to interact with IDM in limited ways without further authentication, such as when a user has not yet logged in and makes a login request. The anonymous user account also allows self-registration.

The default password for the `anonymous` user is `anonymous`.

idm-provisioning

The internal user `idm-provisioning` is a service account used by AM to provision accounts in IDM. It has no password, and isn't meant to be logged in directly. If you are not planning to use AM and IDM together as a platform, you can safely remove this user.

Managed users

Regular user accounts that are stored in IDM's repository are called *managed users* because IDM effectively manages these accounts.

Both internal and managed users *must* authenticate to gain access to the server. The way in which these user types are authenticated is defined in your project's `conf/authentication.json` file.

Any request to IDM will authenticate the user and return a token. To improve tracing through logs, authenticate internal and managed users over REST by sending a POST request to the `openidm/authentication` endpoint, with `_action=login`. The following example authenticates the `openidm-admin` user:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request POST  \
"https://localhost:8443/openidm/authentication?_action=login"
```

Attributes used for authentication

By default, the attribute names that are used to authenticate managed and internal users are `username` and `password`. You can change the attributes that store authentication information with the `propertyMapping` object in the `conf/authentication.json` file. The following excerpt of the `authentication.json` file shows the default authentication attributes:

```
...
    "propertyMapping" : {
        "authenticationId" : "username",
        "userCredential" : "password",
        "userRoles" : "authzRoles"
    },
...
```

If you change the attributes that are used for authentication, you must also change any authentication queries that use those attributes. The following authentication queries are referenced in `authentication.json`:

- `credential-internaluser-query` authenticates internal users.
- `credential-query` authenticates managed users.
- `for-username`

To change the authentication queries for a customized authentication attribute, create a `queryFilters.json` file in your project's `conf` directory. Include the authentication query IDs and the amended query filter, taking into account your changed attributes. The default authentication queries are as follows:

```
{
  "credential-query": {
    "_queryFilter": "/userName eq \"${username}\" AND /accountStatus eq \"active\""
  },
  "credential-internaluser-query": {
    "_queryFilter": "/_id eq \"${username}\""
  },
  "for-userName": {
    "_queryFilter": "/userName eq \"${uid}\""
  }
}
```

The following example `conf/queryFilters.json` file shows the authentication queries adjusted to use the `email` attribute instead of the `username` attribute:

```
{
  "credential-query": {
    "_queryFilter": "/email eq \"${email}\" AND /accountStatus eq \"active\""
  },
  "credential-internaluser-query": {
    "_queryFilter": "/_id eq \"${email}\""
  },
  "for-userName": {
    "_queryFilter": "/email eq \"${uid}\""
  }
}
```

Internal users

Although internal users are considered to be special user accounts, you can manage them over the REST interface as you would any regular user in the repository.

To list the internal users over REST, query the `internal/user` endpoint as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET  \
"http://localhost:8080/openidm/internal/user?_queryFilter=true&fields=_id"
{
  "result": [
    {
      "_id": "openidm-admin",
      "_rev": "00000000ec996921"
    },
    {
      "_id": "anonymous",
      "_rev": "00000000d95a68b1"
    },
    {
      "_id": "idm-provisioning",
      "_rev": "00000000817e3805"
    },
    {
      "_id": "connector-server-client",
      "_rev": "000000003f2a3a85"
    }
  ],
  ...
}
```

To query the details of an internal user, include the user ID in the request, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET  \
"http://localhost:8080/openidm/internal/user/openidm-admin"
{
  "_id": "openidm-admin",
  "_rev": "00000000ec996921"
}
```

Internal users have specific authorization roles by default. These roles determine what the users can access in IDM. The `anonymous` user has only the `openidm-reg` role by default. This role grants only the resource access required to log in, register, and so forth. To identify the authorization roles for the `openidm-admin` internal user, and for information about creating and managing other administrative users, see Administrative Users.

Change the administrator user password

The password of the `openidm-admin` user is `openidm-admin` by default. This password is set in the following excerpt of the `authentication.json` file:

```
{
    "name" : "STATIC_USER",
    "properties" : {
        "queryOnResource" : "internal/user",
        "username" : "openidm-admin",
        "password" : "&{openidm.admin.password}",
        "defaultUserRoles" : [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
        ]
    },
    "enabled" : true
}
```

The `password` property references the `openidm.admin.password` property, set in `resolver/boot.properties`:

```
openidm.admin.password=openidm-admin
```

You can change the default administrator password in a number of ways:

- Edit the `resolver/boot.properties` file before you start IDM (or restart IDM after you change this file).
- Set the value directly in the `conf/authentication.json` file.
- Update the authentication configuration over REST.

    1. Get the current authentication configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/config/authentication"
{
  "_id": "authentication",
  "serverAuthContext": {
    ...
    "authModules": [
      ...
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "openidm-admin",
          "password": "&{openidm.admin.password}",
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      ...
    ]
  }
}
```

2. Change the `password` field of this `STATIC_USER` module and replace the authentication configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request PUT \
--data '{
  "_id": "authentication",
  "serverAuthContext": {
    "sessionModule": {
      "name": "JWT_SESSION",
      "properties": {
        "maxTokenLifeMinutes": 120,
        "tokenIdleTimeMinutes": 30,
        "sessionOnly": true,
        "isHttpOnly": true,
        "enableDynamicRoles": false
      }
    },
    "authModules": [
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "anonymous",
          "password": {
            "$crypto": {
              "type": "x-simple-encryption",
              "value": {
                "cipher": "AES/CBC/PKCS5Padding",
                "stableId": "openidm-sym-default",
                "salt": "xBlTp67ze4Ca5LTocXOpoA==",
                "data": "mdibV6UabU2M+M5MK7bjFQ==",
                "keySize": 16,
                "purpose": "idm.config.encryption",
                "iv": "36D2+FumKbaUsndNQ+/5w==",
                "mac": "ZM8GMnh0n80QwtSH6QsNmA=="
              }
            }
          },
          "defaultUserRoles": [
            "internal/role/openidm-reg"
          ]
        },
        "enabled": true
      },
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "openidm-admin",
          "password": "newAdminPassword",
          "defaultUserRoles": [
```

```
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      {
        "name": "MANAGED_USER",
        "properties": {
          "augmentSecurityContext": {
            "type": "text/javascript",
            "source": "require('auth/customAuthz').setProtectedAttributes(security)"
          },
          "queryId": "credential-query",
          "queryOnResource": "{managed_user}",
          "propertyMapping": {
            "authenticationId": "username",
            "userCredential": "password",
            "userRoles": "authzRoles"
          },
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ]
        },
        "enabled": true
      },
      {
        "name": "SOCIAL_PROVIDERS",
        "properties": {
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ],
          "augmentSecurityContext": {
            "type": "text/javascript",
            "globals": {},
            "file": "auth/populateAsManagedUserFromRelationship.js"
          },
          "propertyMapping": {
            "userRoles": "authzRoles"
          }
        },
        "enabled": true
      }
    ]
  }
}' \
"{secureHostname}/openidm/config/authentication"
{
  "_id": "authentication",
  "serverAuthContext": {
    "sessionModule": {
      "name": "JWT_SESSION",
      "properties": {
        "maxTokenLifeMinutes": 120,
        "tokenIdleTimeMinutes": 30,
```

```
        "sessionOnly": true,
        "isHttpOnly": true,
        "enableDynamicRoles": false
      }
    },
    "authModules": [
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "anonymous",
          "password": {
            "$crypto": {
              "type": "x-simple-encryption",
              "value": {
                "cipher": "AES/CBC/PKCS5Padding",
                "stableId": "openidm-sym-default",
                "salt": "xBlTp67ze4Ca5LTocXOpoA==",
                "data": "mdibV6UabU2M+M5MK7bjFQ==",
                "keySize": 16,
                "purpose": "idm.config.encryption",
                "iv": "36D2+FumKbaUsndNQ/+5w==",
                "mac": "ZM8GMnh0n80QwtSH6QsNmA=="
              }
            }
          },
          "defaultUserRoles": [
            "internal/role/openidm-reg"
          ]
        },
        "enabled": true
      },
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "openidm-admin",
          "password": {
            "$crypto": {
              "type": "x-simple-encryption",
              "value": {
                "cipher": "AES/CBC/PKCS5Padding",
                "stableId": "openidm-sym-default",
                "salt": "l0trJWBzg5JKcWLzNq8QDA==",
                "data": "MKAkL9FVEq/FnWq+8a90+QcjfkEbrK7W4tIc3ORD1ck=",
                "keySize": 16,
                "purpose": "idm.config.encryption",
                "iv": "UMjU6crk332MZtEjo+wEmw==",
                "mac": "7EvTqjpmuS9PmY1aCT2s+g=="
              }
            }
          },
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
```

```
          ]
        },
        "enabled": true
      },
      {
        "name": "MANAGED_USER",
        "properties": {
          "augmentSecurityContext": {
            "type": "text/javascript",
            "source": "require(auth/customAuthz).setProtectedAttributes(security)"
          },
          "queryId": "credential-query",
          "queryOnResource": "managed/user",
          "propertyMapping": {
            "authenticationId": "username",
            "userCredential": "password",
            "userRoles": "authzRoles"
          },
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ]
        },
        "enabled": true
      },
      {
        "name": "SOCIAL_PROVIDERS",
        "properties": {
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ],
          "augmentSecurityContext": {
            "type": "text/javascript",
            "globals": {},
            "file": "auth/populateAsManagedUserFromRelationship.js"
          },
          "propertyMapping": {
            "userRoles": "authzRoles"
          }
        },
        "enabled": true
      }
    ]
  }
}
```

Authentication and session modules

An authentication module specifies how a user or client is authenticated. You configure authentication and session modules in your project's `conf/authentication.json` file.

IDM evaluates authentication modules in the order in which they appear in that file, and uses the first "successful" authentication module it finds. Subsequent modules are not evaluated. In a production environment, you should remove any unused authentication modules from your `authentication.json` file.

To authenticate a user or client, IDM validates the provided credentials against some resource. That resource can be either an IDM resource such as `managed/user` or `internal/user`, or it can be an external resource such as an LDAP server or social identity provider. You should prioritize the authentication modules that query IDM resources over those that query external resources. Prioritizing modules that query external resources can lead to authentication problems for internal users such as `openidm-admin`.

You can also configure authentication modules in the admin UI. Select **Configure > Authentication**, and select the **Session** or **Module** tab. To change the order of authentication modules in the admin UI, simply drag the modules up or down so that they appear in the order in which they should be evaluated.

> **Note**
>
> Modifying an authentication module in the admin UI might affect your current session. In this case, IDM prompts you with the following message:
>
> ```
> Your current session may be invalid. Click here to logout and re-authenticate.
> ```
>
> When you select the **Click here** link, IDM logs you out of any current session and returns you to the login screen.

IDM supports the following authentication and session modules:

JWT_SESSION

IDM supports one session module, the JSON Web Token (JWT) Session Module. When a client authenticates successfully, the JWT Session Module creates a JWT and sets it as a cookie on the response. On subsequent requests, the module checks for the presence of the JWT as a cookie on the request, validates the signature and decrypts it, and checks the expiration time of the JWT.

JWT sessions are entirely stateless, that is, they are not persisted in the backend. All information pertaining to the session is encrypted in the JWT.

When a request to IDM produces a JWT, that value replaces the previous one used to send that request. In this way the JWT is always updated to the latest copy. The idle timeout in the JWT is therefore continuously updated and active sessions are not abruptly killed mid-session.

By default, the JWT cookie is deleted on logout. Deleting the cookie manually ends the session. You can modify what happens to the session after a browser restart by changing the value of the `sessionOnly` property.

The default JWT Session Module configuration, in `conf/authentication.json`, is as follows:

```
"sessionModule" : {
    "name" : "JWT_SESSION",
    "properties" : {
        "maxTokenLifeMinutes" : 120,
        "tokenIdleTimeMinutes" : 30,
        "sessionOnly" : true,
        "isHttpOnly" : true,
        "enableDynamicRoles" : false
    }
}
```

Important

- In a production environment, ensure that only secure cookies are used. To do so, add the following property to the session module configuration:

```
"isSecure" : true
```

- If your `authentication.json` file uses a non-default cookie name for `name.openidm.csrfFilter.authCookieName`, add the following property to the session module configuration:

```
"sessionCookieName": "customCookieName"
```

For more information about this module, refer to the Class JwtSessionModule JavaDoc.

Attempting to access IDM without the appropriate headers or session cookie results in an HTTP 401 Unauthorized, or HTTP 403 Forbidden, depending on the situation. If you authenticate using a session cookie, you must include an additional header that indicates the origin of the request.

The following example shows a successful authentication attempt and the return of a session cookie:

```
curl \
--dump-header /dev/stdout \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
"https://localhost:8443/openidm/managed/user?_queryFilter=true&_fields=_id"
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-cache
Set-Cookie: session-jwt=2l0zobpuk6st1b2m7gvhg5zas ...;Path=/
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Vary: Accept-Encoding, User-Agent
Content-Length: 82
Server: Jetty(8.y.z-SNAPSHOT)
```

The following example uses the cookie returned in the previous response, and includes the `X-Requested-With` header to indicate the origin of the request. The value of the header can be any string, but should be informative for logging purposes. If you do not include the `X-Requested-With` header, IDM returns HTTP 403 Forbidden:

```
curl \
--dump-header /dev/stdout \
--header "Cookie: session-jwt=2l0zobpuk6st1b2m7gvhg5zas ..." \
--header "X-Requested-With: OpenIDM Plugin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
"https://localhost:8443/openidm/managed/user?_queryFilter=true&_fields=_id"
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json; charset=UTF-8
Cache-Control: no-cache
Vary: Accept-Encoding, User-Agent
Content-Length: 82
Server: Jetty(8.y.z-SNAPSHOT)
```

The expiration date of the JWT cookie, January 1, 1970, corresponds to the start of UNIX time. Since that time is in the past, browsers will not store that cookie after the browser session is closed.

Authentication requests are logged in the `authentication.audit.json` file. A successful authentication request is logged as follows:

```
{
  "_id": "389d15d3-bdd5-4521-ae3c-bf096d334405-915",
  "timestamp": "2019-08-02T11:53:31.110Z",
  "eventName": "SESSION",
  "transactionId": "389d15d3-bdd5-4521-ae3c-bf096d334405-912",
  "trackingIds": [
    "5f9f4941-bcbd-4cbc-97f7-e763808e4310",
    "88973bcf-0d60-41b8-9922-73718ce76e11"
  ],
  "userId": "openidm-admin",
  "principal": [
    "openidm-admin"
  ],
  "entries": [
    {
      "moduleId": "JwtSession",
      "result": "SUCCESSFUL",
      "info": {
        "org.forgerock.authentication.principal": "openidm-admin"
      }
    }
  ],
  "result": "SUCCESSFUL",
  "provider": null,
  "method": "JwtSession"
}
```

For information about querying this log, refer to Query the Authentication Audit Log.

Authenticate without a session

Once a client has authenticated, the `JWT_SESSION` takes precedence over any other authentication modules, for subsequent requests. In some cases, you might want to force clients to re-authenticate for each request. This is the case, for example, if you authenticate using a client certificate.

To request one-time authentication without a session, use the `X-OpenIDM-NoSession` header in the authentication request. For example:

```
curl \
--dump-header /dev/stdout \
--header "X-OpenIDM-NoSession: true" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--cacert ca-cert.pem \
--header "Accept-API-Version: resource=1.0" \
"https://localhost:8443/openidm/managed/user?_queryFilter=true&_fields=_id"
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-cache
Vary: Accept-Encoding, User-Agent
Content-Length: 82
Server: Jetty(8.y.z-SNAPSHOT)
```

Deterministic ECDSA signatures

By default, JWTs are signed with deterministic Elliptic Curve Digital Signature Algorithm (ECDSA). In order to use this more secure signing method, Bouncy Castle, which is included in the default IDM installation, must be installed. If Bouncy Castle is unavailable, or the key is incompatible, IDM falls back to normal ECDSA.

> **Note**
>
> If you need to turn off the use of deterministic ECDSA, set the following property in your `conf/system.properties` file:
>
> ```
> org.forgerock.secrets.preferDeterministicEcdsa=false
> ```

STATIC_USER

The `STATIC_USER` module provides an authentication mechanism that avoids database lookups by hard coding a static user. IDM includes a default `anonymous` static user, but you can create any static user for this module.

The following sample REST call uses `STATIC_USER` authentication with the `anonymous` user in the self-registration process:

```
curl \
--header "X-OpenIDM-Password: anonymous" \
--header "X-OpenIDM-Username: anonymous" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "userName": "steve",
  "givenName": "Steve",
  "sn": "Carter",
  "telephoneNumber": "0828290289",
  "mail": "scarter@example.com",
  "password": "Passw0rd"
}' \
"https://localhost:8443/openidm/managed/user/?_action=create"
```

Note

This is not the same as an anonymous request that is issued without headers.

Authenticating with the `STATIC_USER` module avoids the performance cost of reading the database for self-registration, certain UI requests, and other actions that can be performed anonymously. Authenticating the anonymous user with the `STATIC_USER` module is identical to authenticating the anonymous user with the `INTERNAL_USER` module, except that the database is not accessed. So, `STATIC_USER` authentication provides an authentication mechanism for the anonymous user that avoids the database lookups incurred when using `INTERNAL_USER`.

A sample `STATIC_USER` authentication configuration follows:

```
{
    "name" : "STATIC_USER",
    "enabled" : true,
    "properties" : {
        "queryOnResource" : "internal/user",
        "username" : "anonymous",
        "password" : "anonymous",
        "defaultUserRoles" : [
            "internal/role/openidm-reg"
        ]
    }
}
```

IDM also uses the `STATIC_USER` module to set the password and default roles of the `openidm-admin` internal user on startup. The following configuration in the `authentication.json` file sets up the `openidm-admin` user:

```
{
    "name" : "STATIC_USER",
    "properties" : {
        "queryOnResource" : "internal/user",
        "username" : "openidm-admin",
        "password" : "&{openidm.admin.password}",
        "defaultUserRoles" : [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
        ]
    },
    "enabled" : true
}
```

Related topics:

- Change the administrator user password (`openidm-admin`)
- Secure access to RCS

TRUSTED_ATTRIBUTE

The `TRUSTED_ATTRIBUTE` authentication module lets you configure IDM to trust a specific `HttpServletRequest` attribute. To enable this module, add it to your `authentication.json` file as follows:

```
{
    "name" : "TRUSTED_ATTRIBUTE",
    "properties" : {
        "queryOnResource" : "managed/user",
        "propertyMapping" : {
            "authenticationId" : "userName",
            "userRoles" : "authzRoles"
        },
        "defaultUserRoles" : [ ],
        "authenticationIdAttribute" : "X-ForgeRock-AuthenticationId",
        "augmentSecurityContext" : {
            "type" : "text/javascript",
            "file" : "auth/populateRolesFromRelationship.js"
        }
    },
    "enabled" : true
}
```

`TRUSTED_ATTRIBUTE` authentication queries the `managed/user` resource, and allows authentication when credentials match, based on the `username` and `authzRoles` assigned to that user, specifically the `X-ForgeRock-AuthenticationId` attribute.

Tip

To use the `TRUSTED_ATTRIBUTE` module with internal `authz` roles, you must modify the `isAJAXRequest` function in `bin/defaults/script/router-authz.js` to check for the `X-Special-Trusted-User` header:

```
function isAJAXRequest() {
   var headers = context.http.headers;
   // one of these custom headers must be present for all HTTP-based requests, to prevent CSRF attacks

   // X-Requested-With is common from AJAX libraries such as jQuery
   if (typeof (headers["X-Requested-With"]) !== "undefined" ||
       typeof (headers["x-requested-with"]) !== "undefined" ||

       // Basic auth headers are acceptible for convenience from cURL commands;
       // We don't return the request header to prompt the browser to provide basic auth headers,
       // so it will only be present if someone explicitly provides them, as in a cURL request.
       typeof (headers["Authorization"]) !== "undefined" ||
       typeof (headers["authorization"]) !== "undefined" ||

       // The custom authn headers for OpenIDM
       typeof (headers["X-OpenIDM-Username"]) !== "undefined" ||
       typeof (headers["x-openidm-username"]) !== "undefined" ||
       typeof (headers["X-Special-Trusted-User"]) !== "undefined" ||
       typeof (headers["x-special-trusted-user"]) !== "undefined") {

       if ((headers["X-Requested-With"] || "").toLowerCase().startsWith("shockwaveflash")) {
           // prevent CSRF from Flash
           return false;
       }
       return true;
   }
   return false;
}
```

For a sample implementation of a custom servlet filter and the Trusted Request Attribute Authentication Module, refer to Authenticate using a trusted servlet filter.

MANAGED_USER

`MANAGED_USER` authentication queries the repository and allows authentication if the credentials match. Despite the module name, the query is not restricted to managed/user objects. The resource that is queried is configurable. The default configuration uses the `username` and `password` of a managed user to authenticate, as shown in the following sample configuration:

```
{
    "name" : "MANAGED_USER",
    "properties" : {
        "augmentSecurityContext": {
            "type" : "text/javascript",
            "source" : "require('auth/customAuthz').setProtectedAttributes(security)"
        },
        "queryId" : "credential-query",
        "queryOnResource" : "managed/user",
        "propertyMapping" : {
            "authenticationId" : "username",
            "userCredential" : "password",
            "userRoles" : "authzRoles"
        },
        "defaultUserRoles" : [
            "internal/role/openidm-authorized"
        ]
    },
    "enabled" : true
}
```

Use the `augmentSecurityContext` property to add custom properties to the security context of users who authenticate with this module. By default, this property adds a list of *protected properties* to the user's security context. These protected properties are defined in the managed object schema. The `isProtected` property is described in Create and modify object types.

INTERNAL_USER

`INTERNAL_USER` authentication queries the `internal/user` objects in the repository and allows authentication if the credentials match. An example configuration that uses the `username` and `password` of the internal user to authenticate follows:

```
{
    "name" : "INTERNAL_USER",
    "enabled" : true,
    "properties" : {
        "queryId" : "credential-internaluser-query",
        "queryOnResource" : "internal/user",
        "propertyMapping" : {
            "authenticationId" : "username",
            "userCredential" : "password",
            "userRoles" : "authzRoles"
        },
        "defaultUserRoles" : [ ]
    }
}
```

CLIENT_CERT

Client certificate authentication (also called *mutual SSL authentication*) occurs as part of the SSL or TLS handshake, which takes place before any data is transmitted in an SSL or TLS session. This authentication module is typically used when users have secure certificates that they install in their browsers for authentication and authorization.

The client certificate module, `CLIENT_CERT`, authenticates by validating a client certificate, transmitted through an HTTP request. IDM compares the subject DN of the request certificate with the subject DN of the truststore.

A sample `CLIENT_CERT` authentication configuration follows:

```
{
  "name" : "CLIENT_CERT",
  "properties" : {
    "augmentSecurityContext" : {
      "type" : "text/javascript",
      "globals" : { },
      "file" : "auth/mapUserFromClientCert.js"
    },
    "queryOnResource" : "managed/user",
    "defaultUserRoles" : [
      "internal/role/openidm-authorized"
    ],
    "allowedAuthenticationIdPatterns" : [
      ".*CN=localhost, O=ForgeRock.*"
    ]
  },
  "enabled" : true
}
```

When a user authenticates with a client certificate, they receive the roles listed in the `defaultUserRoles` property of the `CLIENT_CERT` module. Privileges are calculated dynamically per request when enabled in the session module.

> **Note**
>
> Client certificate authentication is also used when the client is a password plugin, such as those described in Password Plugins. This process is similar to an administrative request to modify the passwords of regular users. For password plugin clients, you must include `internal/role/openidm-cert` in the `defaultUserRoles` array (in the authentication configuration).

Test client certificate authentication

This procedure demonstrates client certificate authentication by generating a self-signed certificate, adding that certificate to the truststore, then authenticating with the certificate. At the end of this procedure, you will verify the certificate over port `8444` as defined in your project's `resolver/boot.properties` file:

```
openidm.auth.clientauthonlyports=8444
```

The example assumes an existing managed user, bjensen, with email address `bjensen@example.com`.

1. Create a self-signed certificate for user bjensen as follows:

```
openssl req \
-x509 \
-newkey rsa:1024 \
-keyout /path/to/key.pem \
-out /path/to/cert.pem \
-days 3650 \
-nodes
Generating a 1024 bit RSA private key
..................
..................
writing new private key to 'key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []: US
State or Province Name (full name) []: Washington
Locality Name (eg, city) []: Vancouver
Organization Name (eg, company) []: Example.com
Organizational Unit Name (eg, section) []:
Common Name (eg, fully qualified host name) []: localhost
Email Address []: bjensen@example.com
```

> **Note**
>
> The `Email Address` is used by the `mapUserFromClientCert.js` to map the user against an existing managed user.

2. Import the client certificate into the IDM truststore:

```
keytool \
-importcert \
-keystore /path/to/openidm/security/truststore \
-storetype JKS \
-storepass changeit \
-file /path/to/cert.pem \
-trustcacerts \
-noprompt \
-alias client-cert-example
Certificate was added to keystore
```

By default, users can authenticate only if their certificates have been issued by a Certificate Authority (CA) that is listed in the truststore. The default truststore includes several trusted root CA certificates, and any user certificate issued by those CAs will be trusted. Change the value of this property to restrict certificates to those issued to users in your domain, or use some other regular expression to limit who will be trusted. If you leave this property empty, no certificates will be trusted.

3. Edit your project's `conf/authentication.json` file. Add the `CLIENT_CERT` module, and add at least the email address from the certificate subject DN to the `allowedAuthenticationIdPatterns`:

```
{
  "name": "CLIENT_CERT",
  "properties": {
    "augmentSecurityContext": {
      "type": "text/javascript",
      "globals": {},
      "file": "auth/mapUserFromClientCert.js"
    },
    "queryOnResource": "managed/user",
    "defaultUserRoles": [
      "internal/role/openidm-cert",
      "internal/role/openidm-authorized"
    ],
    "allowedAuthenticationIdPatterns": [
      ".*EMAILADDRESS=bjensen@example.com.*"
    ]
  },
  "enabled": true
}
```

Note

The `allowedAuthenticationIdPatterns` property is unique to this authentication module. This property contains a regular expression that defines which user distinguished names (DNs) are allowed to authenticate with a certificate.

4. Send an HTTP request with your certificate file `cert.pem` to the secure port:

```
curl \
--insecure \
--cert-type PEM \
--key /path/to/key.pem \
--key-type PEM \
--cert /path/to/cert.pem \
--header "X-Requested-With: curl" \
--header "X-OpenIDM-NoSession: true" \
--request GET "https://localhost:8444/openidm/info/login"
{
  "_id": "login",
  "authenticationId": "EMAILADDRESS=bjensen@example.com, CN=localhost, O=Example.com, L=Vancouver,
ST=Washington, C=US",
  "authorization": {
    "userRolesProperty": "authzRoles",
    "component": "managed/user",
    "authLogin": false,
    "roles": [
      "internal/role/openidm-cert",
      "internal/role/openidm-authorized"
    ],
    "ipAddress": "0:0:0:0:0:0:0:1",
    "id": "aba3e666-c0db-4669-8760-0eb21f310649",
    "moduleId": "CLIENT_CERT"
  }
}
```

Note

- Because we have used a self-signed certificate in this example, you must include the `--insecure` option. You should not include this option if you are using a CA cert.
- You must use the `X-Requested-With` and `X-OpenIDM-NoSession` headers for HTTP-based requests that use the `CLIENT_CERT` authentication module.

PASSTHROUGH

`PASSTHROUGH` authentication queries an external system, such as an LDAP server, and allows authentication if the credentials included in the REST request match those in the external system.

The following excerpt of an `authentication.json` shows a pass-through authentication configuration for an LDAP system:

```
"authModules" : [
    {
       "name" : "PASSTHROUGH",
       "enabled" : true,
       "properties" : {
          "augmentSecurityContext": {
             "type" : "text/javascript",
             "file" : "auth/populateAsManagedUser.js"
          },
          "queryOnResource" : "system/ldap/account",
          "propertyMapping" : {
             "authenticationId" : "uid",
             "groupMembership" : "ldapGroups"
          },
          "groupRoleMapping" : {
             "internal/role/openidm-admin" : ["cn=admins,ou=Groups,dc=example,dc=com"]
          },
          "defaultUserRoles" : [
             "internal/role/openidm-authorized"
          ]
       },
    },
    ...
]
```

For more information on authentication module properties, refer to Authentication and session module configuration.

Many of the documented samples are configured for pass-through authentication. For example, the `sync-with-ldap*` samples use an external LDAP system for authentication.

SOCIAL_PROVIDERS

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

The `SOCIAL_PROVIDERS` module enables authentication through social identity providers that comply with OAuth 2.0 and OpenID Connect 1.0 standards.

For information about configuring this module with social identity providers such as Google, LinkedIn, and Facebook, refer to Social providers authentication module.

The `/path/to/openidm/audit/authentication.audit.json` log file shows the corresponding `SOCIAL_AUTH` module, used to handle authentication for each social identity provider.

IWA

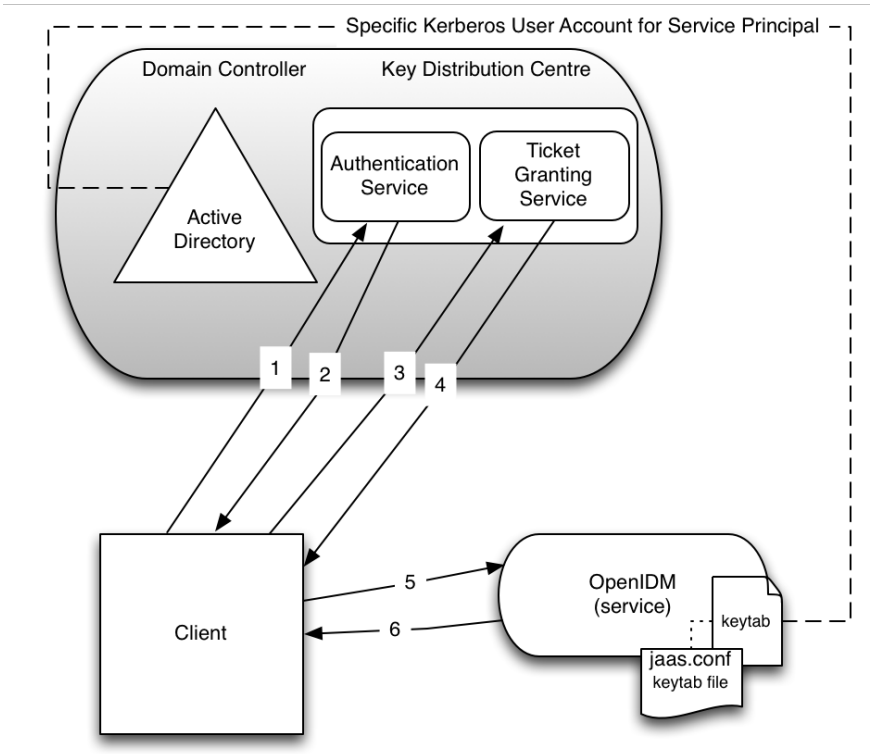
The `IWA` module lets users authenticate using Integrated Windows Authentication (IWA) with Kerberos instead of a username and password.

Windows, UNIX, and Linux systems support Kerberos v5 authentication, which can operate safely on an open, unprotected network. With Kerberos authentication, the user or client application obtains temporary credentials for a service from an *authorization server*, in the form of tickets and session keys. The *service server* handles its part of the Kerberos mutual authentication process.

To enable Kerberos authentication, IDM requires a specific Kerberos user account in Active Directory, and a keytab file that maps the service principal to this user account. The client presents IDM with a Kerberos ticket. If IDM can validate the ticket, the client is granted an encrypted session key for the IDM service. That client can then access IDM without providing a username or password, for the duration of the session.

The complete Kerberos authentication process is shown in the following diagram:

Client Authentication to IDM Using a Kerberos Ticket



1 - Client requests TGT from KDC
2 - Authentication service sends encrypted TGT and session key
3 - Client requests server access from TGS
4 - TGC sends encrypted session key and ticket

This section assumes that you have an active Kerberos server acting as a Key Distribution Center (KDC). If you are running Active Directory, that service includes a Kerberos KDC by default.

Create a specific Kerberos user account

To authenticate IDM to the Kerberos KDC you must create a specific user entry in Active Directory whose credentials will be used for this authentication. This Kerberos user account must not be used for anything else.

The Kerberos user account is used to generate the Kerberos keytab. If you change the password of this Kerberos user after you have set up IWA, you must update the keytab accordingly.

Create a new user in Active Directory as follows:

1. Select **New > User** and provide a login name for the user that reflects its purpose; for example, **openidm@example.com**.
2. Enter a password for the user. Select the **Password never expires** checkbox, and leave all other options unselected.

If the password of this user account expires, and is reset, you must update the keytab with the new password. It is therefore easier to create an account with a password that does not expire.

3. To create the user, click **Finish**.

Create a keytab file

A Kerberos keytab file (`krb5.keytab`) enables IDM to validate the Kerberos tickets that it receives from client browsers. You must create a Kerberos keytab file for the host on which IDM is running.

This section describes the `ktpass` command, included in the Windows Server toolkit, to create the keytab file. Run the `ktpass` command on the Active Directory domain controller.

Important

- The keytab file is case-sensitive, use the capitalization in this example.
- You must disable UAC or run the `ktpass` command as a user with administrative privileges.

The following command creates a keytab file (named `openidm.HTTP.keytab`) for the IDM service located at `openidm.example.com`.

```
C:\Users\Administrator>ktpass ^
-princ HTTP/openidm.example.com@EXAMPLE.COM ^
-mapUser EXAMPLE\openidm ^
-mapOp set ^
-pass Passw0rd1 ^
-crypto ALL
-pType KRB5_NT_PRINCIPAL ^
-kvno 0 ^
-out openidm.HTTP.keytab
Targeting domain controller: host.example.com
Using legacy password setting method
Successfully mapped HTTP/openidm.example.com to openidm.
Key created.
Output keytab to openidm.HTTP.keytab:
Keytab version: 0x502
keysize 79 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x1 (DES-CBC-CRC) keylength 8 (0x73a28fd307ad4f83)
keysize 79 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x3 (DES-CBC-MD5) keylength 8 (0x73a28fd307ad4f83)
keysize 87 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x17 (RC4-HMAC) keylength 16 (0xa87f3a337d73085c45f9416be5787d86)
keysize 103 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x12 (AES256-SHA1) keylength 32 (0x6df9c282abe3be787553f23a3d1fcefc
  6fc4a29c3165a38bae36a8493e866d60)
keysize 87 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x11 (AES128-SHA1) keylength 16 (0xf616977f071542cd8ef3ff4e2ebcc09c)
```

The `ktpass` command takes the following options:

- `-princ` specifies the service principal name in the format `service/host-name@realm`

In this example (`HTTP/openidm.example.com@EXAMPLE.COM`), the client browser constructs an SPN based on the following:

- The service name (HTTP).

  The service name for SPNEGO web authentication *must* be HTTP.

- The FQDN of the host on which IDM runs (`openidm.example.com`).

  This example assumes that users will access IDM at the URL `https://openidm.example.com:8443`.

- The Kerberos realm name (`EXAMPLE.COM`).

  The realm name must be uppercase. A Kerberos realm defines the area of authority of the Kerberos authentication server.

- `-mapUser` specifies the name of the Kerberos user account to which the principal should be mapped (the account that you created in Creating a Specific Kerberos User Account). The username must be specified in down-level logon name format (DOMAIN\UserName). In our example, the Kerberos user name is `EXAMPLE\openidm`.
- `-mapOp` specifies how the Kerberos user account is linked. Use `set` to set the first user name to be linked. The default (`add`) adds the value of the specified local user name if a value already exists.
- `-pass` specifies a password for the principal user name. Use `*` to prompt for a password.
- `-crypto` specifies the cryptographic type of the keys that are generated in the keytab file. Use `ALL` to specify all crypto types.

  This procedure assumes a 128-bit cryptosystem, with a default RC4-HMAC-NT cryptography algorithm. You can use the `ktpass` command to view the crypto algorithm, as follows:

```
C:\Users\Administrator>ktpass -in .\openidm.HTTP.keytab
Existing keytab:
Keytab version: 0x502
keysize 79 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x1 (DES-CBC-CRC) keylength 8 (0x73a28fd307ad4f83)
keysize 79 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x3 (DES-CBC-MD5) keylength 8 (0x73a28fd307ad4f83)
keysize 87 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x17 (RC4-HMAC) keylength 16 (0xa87f3a337d73085c45f9416be5787d86)
keysize 103 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x12 (AES256-SHA1) keylength 32 (0x6df9c282abe3be787553f23a3d1fcefc6
 fc4a29c3165a38bae36a8493e866d60)
keysize 87 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x11 (AES128-SHA1) keylength 16 (0xf616977f071542cd8ef3ff4e2ebcc09c)
```

- `-ptype` Specifies the principal type. Use `KRB5_NT_PRINCIPAL`.
- `-kvno` specifies the key version number. Set the key version number to 0.
- `-out` specifies the name of the keytab file that will be generated, for example, `openidm.HTTP.keytab`.

> **Note**
>
> The keys that are stored in the keytab file are similar to user passwords. You must protect the Kerberos keytab file in the same way that you would protect a file containing passwords.

For more information about the `ktpass` command, refer to the ktpass reference in the Windows server documentation.

Configure IDM for IWA

To configure the IWA authentication module, add the module to your project's `conf/authentication.json` file.

This section assumes that the connection from IDM to the Active Directory Server is through an LDAP connector, and that the mapping from managed users to the users in Active Directory (in your project's `conf/sync.json` file) identifies the Active Directory target as `system/ad/account`. If you have named the target differently, modify the `"queryOnResource" : "system/ad/account"` property accordingly.

Add the IWA authentication module towards the end of your `conf/authentication.json` file. For example:

```
"authModules" : [
    ...
    {
        "name": "IWA",
        "properties": {
            "servicePrincipal": "HTTP/openidm.example.com@EXAMPLE.COM",
            "keytabFileName": "C:\\Users\\Administrator\\openidm\\security\\openidm.HTTP.keytab",
            "kerberosRealm": "EXAMPLE.COM",
            "kerberosServerName": "kdc.example.com",
            "queryOnResource": "system/ad/account",
            "maxTokenSize": 48000,
            "propertyMapping": {
                "authenticationId": "sAMAccountName",
                "groupMembership": "memberOf"
            },
            "groupRoleMapping": {
                "internal/role/openidm-admin": [ ]
            },
            "groupComparisonMethod": "ldap",
            "defaultUserRoles": [
                "internal/role/openidm-authorized"
            ],
            "augmentSecurityContext": {
                "type": "text/javascript",
                "file": "auth/populateAsManagedUser.js"
            }
        },
        "enabled": true
    },
    ...
]
```

The IWA authentication module includes the following configurable properties:

`servicePrincipal`

The Kerberos principal for authentication, in the following format:

`HTTP/host.domain@DC-DOMAIN-NAME`

host and domain correspond to the host and domain names of the IDM server. DC-DOMAIN-NAME is the domain name of the Windows Kerberos domain controller server. The DC-DOMAIN-NAME can differ from the domain name for the IDM server.

`keytabFileName`

The full path to the keytab file for the Service Principal. On Windows systems, any backslash (`\`) characters in the path must be escaped, as shown in the previous example.

`kerberosRealm`

The Kerberos Key Distribution Center realm. For the Windows Kerberos service, this is the domain controller server domain name.

`kerberosServerName`

The fully qualified domain name of the Kerberos Key Distribution Center server, such as that of the domain controller server.

`queryOnResource`

The IDM resource to check for the authenticating user; for example, `system/ad/account`.

`maxTokenSize`

During the Kerberos authentication process, the Windows server builds a token to represent the user for authorization. This property sets the maximum size of the token, to prevent DoS attacks, if the SPENGO token in the request being made is amended with extra data. The default maximum token size is `48000` bytes.

`groupRoleMapping`

Enables you to grant different roles to users who are authenticated through the `IWA` module.

You can use the `IWA` module in conjunction with the `PASSTHROUGH` authentication module. In this case, a failure in the `IWA` module lets users revert to forms-based authentication.

To add the `PASSTHROUGH` module, follow PASSTHROUGH.

Authenticate through AM

> **Important**
>
> From IDM 7.0 onwards, using AM bearer tokens for authentication is the only supported method of integrating IDM with AM.

ForgeRock Access Management (AM) is a ForgeRock product that provides an infrastructure for managing users, roles, and access to resources. When you use IDM and AM together in a *platform deployment*, you must configure IDM to use AM bearer tokens for authentication. This delegates all authentication to AM. In this configuration:

- IDM uses an `rsFilter` that replaces all other authentication methods.

- All IDM endpoints that require authentication are accessed using an authorization header that contains the bearer token, instead of `X-OpenIDM-Username` and `X-OpenIDM-Password`. Endpoints that allow anonymous access can be accessed without a token.

To use AM bearer tokens for authentication, your AM configuration must include at least the following configuration:

- Two OAuth 2.0 clients: an `idm-resource-server` client to introspect the bearer token, and an `idm-provisioning` client used by AM to provision users in IDM. For information on configuring these clients, refer to Configure OAuth Clients (separate identity stores) or Configure OAuth Clients (shared identity store) in the *Platform Setup Guide*.
- An OAuth 2 provider service.
- An IDM provisioning service.

Your IDM authentication configuration must include the `rsFilter` configuration and *no other* authentication methods.

Sample `rsFilter` configuration

The following sample `rsFilter` configuration is also available in `/path/to/openidm/samples/example-configurations/conf/rsfilter/authentication.json`:

```
{
    "rsFilter" : {
        "clientId" : "",
        "clientSecret" : "",
        "tokenIntrospectUrl" : "http://am.example:8080/openam/oauth2/introspect",
        "scopes" : [ ],
        "cache" : {
            "maxTimeout" : "300 seconds"
        },
        "augmentSecurityContext" : {
            "type" : "text/javascript",
            "source" : "require('auth/orgPrivileges').assignPrivilegesToUser(resource, security, properties,
subjectMapping, privileges, 'privileges', 'privilegeAssignments');"
        },
        "subjectMapping" : [
            {
                "resourceTypeMapping" : {
                    "usr" : "managed/user"
                },
                "propertyMapping" : {
                    "sub" : "_id"
                },
                "userRoles" : "authzRoles/*",
                "additionalUserFields" : [
                    "adminOfOrg",
                    "ownerOfOrg"
                ],
                "defaultRoles" : [
                    "internal/role/openidm-authorized"
                ]
            }
        ],
        "staticUserMapping" : [
            {
                "subject" : "(usr!amadmin)",
                "localUser" : "internal/user/openidm-admin",
                "roles" : [
                    "internal/role/openidm-authorized",
                    "internal/role/openidm-admin"
                ]
            },
            {
                "subject" : "(age!idm-provisioning)",
                "localUser" : "internal/user/idm-provisioning",
                "roles" : [
                    "internal/role/platform-provisioning"
                ]
            }
        ],
        "anonymousUserMapping" : {
            "localUser" : "internal/user/anonymous",
            "roles" : [
                "internal/role/openidm-reg"
            ],
            "executeAugmentationScript" : false
        }
    }
}
```

The `rsFilter` configuration includes the following properties:

`clientId`

The client ID of the AM OAuth 2.0 client used to introspect the bearer token (the `idm-resource-server`) client, in this example).

`clientSecret`

The client secret of the AM OAuth 2.0 client used to introspect the bearer token. IDM encrypts this field if it isn't already.

`tokenIntrospectUrl`

The URI to reach the `oauth2/introspect` endpoint in AM; for example, `http://am.example:8080/openam/oauth2/introspect`.

`scopes`

Any scopes required to be present in the access token. This varies depending on your configuration. For more information about scopes, refer to OAuth 2.0 Scopes in the AM *OAuth 2.0 Guide*.

`cache`

Sets the `maxTimeout`, in seconds, after which the token is removed from the cache.

`augmentSecurityContext`

Specifies a script that is executed only after a successful authentication request. The script helps to populate the expected security context. For more information, refer to The `augmentSecurityContext` trigger.

> **Tip**
>
> You can disable execution of the script for anonymous and static user mappings using the `executeAugmentationScript` property. Doing so can improve IDM's performance for user mappings where it's not necessary to run the script.

`subjectMapping`

An array of mappings that let you map AM realms to IDM managed object types. For example:

```
"subjectMapping" : [
    {
        "resourceTypeMapping" : {
            "usr" : "managed/user"
        },
        "propertyMapping" : {
            "sub" : "_id"
        },
        "userRoles" : "authzRoles/*",
        "additionalUserFields" : [
            "adminOfOrg",
            "ownerOfOrg"
        ],
        "defaultRoles" : [
            "internal/role/openidm-authorized"
        ]
    }
],
```

Each `subjectMapping` includes the following properties:

- Either a `resourceTypeMapping` or a `queryOnResource` property:
  - `resourceTypeMapping`: Maps the identity type of a subject claim in AM to a resource collection in IDM. In the access token, the subject claim is a compound identity that consists of the claim `type` and subject name, separated by a `!`.

    To use a `resourceTypeMapping`, unique Oauth2 subject claims must be enabled in AM. (From AM 7.1, these are enabled by default.) For more information about subject claims, refer to *About the Subject and the Subname Claims* in the section on /oauth2/userinfo.
  - `queryOnResource`: The IDM resource to check for the authenticating user; for example, `managed/user`.

    Tip

    If your AM and IDM deployments use consistent realm and managed object naming, you can configure the `resourceTypeMapping` and the `queryOnResource` properties to let a single subject mapping match multiple realms. This uses a dynamic handlebars template, as in the following example:

    Example configuration

    ```
    "resourceTypeMapping" : {
        "usr" : "managed/{{substring realm 1}}"
    }
    ```

    This configuration lets an access token with the realm `employee` map to an IDM `managed/employee`, and an access token with the realm `contractor` map to an IDM `managed/contractor`.
- `realm`: The AM realm to which this subject mapping applies. A value of `/` specifies the top-level realm. If this property is absent, the mapping can apply to any realm, which is useful if the `resourceTypeMapping` or `queryOnResource` uses a dynamic handlebars template.

You cannot have more than one mapping for the same realm, and you cannot have more than one mapping that has no `realm` in the configuration.

- `propertyMapping`: Maps fields in the AM access token to properties in IDM. This mapping is used to construct the query filter that locates the authenticating user. The default configuration maps the subject (`sub`) in the access token to the `_id` in IDM.
- `userRoles`: Determines the field to consult for locating the authorization roles; usually `authzRoles`, unless you have changed how user roles are stored. This field must be a relationship field. IDM uses the `_refId` from the array elements to populate the user roles in the security context.
- `additionalUserFields`: Determines the field to consult for locating the authorization roles; usually `authzRoles`, unless you have changed how user roles are stored. This field must be a relationship field. IDM uses the `_refId` from the array elements to populate the user roles in the security context.
- `defaultRoles`: The default roles that should be applied to a user who authenticates using the `rsFilter`.

Although you can configure an array of subject mappings, only one mapping is selected and used during the authentication process. If there is a `realm` attribute in the access token, that realm is used to select an appropriate mapping. If no mapping is defined for the access token's realm, or if the realm is not provided in the access token, the authentication uses a mapping that does not define a `realm`.

Subject mapping with a Remote Connector Server (RCS)

If you have an RCS that is authenticating against AM, you must add a subject mapping for it.

Example configuration

```
{
    "subject" : "RCS-OAuth-clientId",
    "localUser" : "internal/user/idm-provisioning"
}
```

The `subject` property must be the OAuth2 client in AM set up for the remote connector server. The `localUser` property can be any existing user.

> **Warning**
>
> Do not assign the `localUser` any roles. Doing so can allow the RCS bearer token to be misused.

`staticUserMapping`

Maps AM users to a matching IDM user. Can contain multiple user mappings, each with the following properties:

- `subject` of the access token (the AM user). If you have specified a `resourceTypeMapping`, the static user mapping includes the full new subject string to match service accounts or static subject mappings, for example:

```
"subject" : "(usr!amadmin)"
```

- `localUser` is the IDM user you want to associate with the AM user identified in `subject`. For example, if `subject` is set to `(usr!amadmin)`, you can set the corresponding `localUser` to `internal/user/openidm-admin`.
- `roles` the default IDM roles that this mapped user has after they authenticate.
- `executeAugmentationScript`: a boolean value. When `false`, the scripts specified in `augmentSecurityContext` will not run for the user mapping.

> **Important**
>
> The `idm-provisioning` subject is a service account used by AM to provision users in IDM. You *must* include this subject in your `staticUserMapping`, for example:
>
> ```
> {
>     "subject": "(age!idm-provisioning)",
>     "localUser": "internal/user/idm-provisioning",
>     "roles" : [
>         "internal/role/platform-provisioning"
>     ]
> }
> ```

`anonymousUserMapping`

The default user used when no access token is included in the request. Contains the following:

- `localUser`: the IDM user resource referenced when no specific user is identified. For example, `internal/user/anonymous`.
- `roles`: the default roles the anonymous user has, usually `internal/role/openidm-reg`.
- `executeAugmentationScript`: a boolean value. When `false`, the scripts specified in `augmentSecurityContext` will not run for the user mapping.

Test authentication through AM

1. Obtain a bearer token from AM. For example:

```
curl \
--header "X-OpenAM-Username: amAdmin" \
--header "X-OpenAM-Password: password" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--request POST \
--data "grant_type=client_credentials" \
--data "client_id=idm-provisioning" \
--data "client_secret=openidm" \
--data "scope=fr:idm:*" \
"http://am.example.com:8080/am/oauth2/realms/root/access_token"
{
  "access_token": "z4uKDWiv4wnxKY7OjeG04PujG8E",
  "scope": "fr:idm:*",
  "token_type": "Bearer",
  "expires_in": 3599
}
```

2. Authenticate to IDM using that bearer token:

```
curl \
--request GET \
--header "Content-Type: application/json" \
--header "Authorization: Bearer z4uKDWiv4wnxKY7OjeG04PujG8E" \
'http://localhost:8080/openidm/info/login'
{
  "_id": "login",
  "authenticationId": "idm-provisioning",
  "authorization": {
    "id": "idm-provisioning",
    "roles": [
      "internal/role/platform-provisioning"
    ],
    "component": "internal/user"
  }
}
```

Refer to the Platform Setup Guide for complete instructions on setting up IDM to use AM bearer tokens for authentication.

Authenticate as a different user

The `X-OpenIDM-RunAs` header lets an administrative user *masquerade* as a regular user, without needing that user's password. To support this header, you must add the `runAsProperties` object to the `properties` of your authentication module. For example:

```
"runAsProperties" : {
    "adminRoles" : [
        "internal/role/openidm-admin"
    ],
    "disallowedRunAsRoles" : [
        "internal/role/openidm-admin"
    ],
    "defaultUserRoles" : [
        "internal/role/openidm-authorized"
    ],
    "queryId" : "credential-query",
    "queryOnResource" : "managed/user",
    "propertyMapping" : {
        "authenticationId" : "username",
        "userRoles" : "authzRoles"
    },
    "augmentSecurityContext" : {
        "type" : "text/javascript",
        "source" : "require('auth/customAuthz').setProtectedAttributes(security)"
    }
}
```

This configuration lets a user who authenticates with the `openidm-admin` role masquerade as any user *except* one with the `openidm-admin` role.

If you are adding this configuration to the `STATIC_USER` module, and you are using Delegated administration, you must add an additional `propertyMapping` to the `properties` of the authentication module. This mapping forces the privileges to be re-read into the security context when the session JWT is used on subsequent requests. For example:

```
"propertyMapping" : {
    "authenticationId" : "username"
}
```

The sample `authentication.json` file in `openidm/samples/example-configurations/conf/runas/` adds the `runAsProperties` object to the `STATIC_USER` module. Users or clients who authenticate with this module can then masquerade as other users.

In the following example, the `openidm-admin` user authenticates with the `STATIC_USER` module, and can run REST calls as user `bjensen` without that user's password:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "X-OpenIDM-RunAs: bjensen" \
--request GET \
"http://localhost:8080/openidm/info/login"
{
  "_id": "login",
  "authenticationId": "bjensen",
  "authorization": {
    "userRolesProperty": "authzRoles",
    "component": "managed/user",
    "authLogin": false,
    "adminUser": "openidm-admin",
    "roles": [
      "internal/role/openidm-authorized"
    ],
    "ipAddress": "[0:0:0:0:0:0:0:1]",
    "authenticationId": "openidm-admin",
    "protectedAttributeList": [
      "password"
    ],
    "id": "bjensen",
    "moduleId": "STATIC_USER",
    "queryId": "credential-query"
  }
}
```

The authentication output shows that the request was made as user `bjensen` but with an `adminUser` of `openidm-admin`. This information is also logged in the authentication audit log.

If you were to actually authenticate as user `bjensen`, without the `runAs` header, the user is authenticated with the `MANAGED_USER` authentication module. The output still shows an `authenticationId` of `bjensen` but there is no reference to an `adminUser`:

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/info/login"
{
  "_id": "login",
  "authenticationId": "bjensen",
  "authorization": {
    "userRolesProperty": "authzRoles",
    "component": "managed/user",
    "authLogin": false,
    "authenticationIdProperty": "username",
    "roles": [
      "internal/role/openidm-authorized"
    ],
    "ipAddress": "[0:0:0:0:0:0:0:1]",
    "authenticationId": "bjensen",
    "protectedAttributeList": [
      "password"
    ],
    "id": "bjensen",
    "moduleId": "MANAGED_USER",
    "queryId": "credential-query"
  }
}
```

Authentication and roles

When a user authenticates, they are given a set of default *internal roles*. These roles determine how much access the user has to IDM. IDM includes a number of default internal roles, on the `openidm/internal/roles` endpoint. You can configure additional internal roles to customize how you restrict access to the server.

The following internal roles are defined by default (in `conf/repo.init.json`):

The following internal roles are defined by default:

openidm-admin

IDM administrator role, excluded from the reauthorization required policy definition by default.

openidm-authorized

Default role for any user who authenticates with a username and password.

openidm-cert

Default role for any user who authenticates with mutual SSL authentication.

This role applies only to mutual authentication. The shared secret (certificate) must be adequately protected. The `openidm-cert` role is excluded from the reauthorization required policy definition by default.

openidm-reg

Assigned to users who access IDM with the default anonymous account.

The `openidm-reg` role is excluded from the reauthorization required policy definition by default.

openidm-tasks-manager

Role for users who can be assigned to workflow tasks.

platform-provisioning

Role for platform provisioning access. If you are not planning to run AM and IDM together as a platform, you can safely remove this role.

When a user authenticates, IDM calculates that user's roles as follows:

- Each authentication module includes a `defaultUserRoles` property. Depending on how the user authenticates, IDM assigns the roles listed in that module's `defaultUserRoles` property to the user on authentication. The `defaultUserRoles` property is specified as an array. For most authentication modules, the user obtains the `openidm-authorized` role on authentication. For example:

```
{
    "name" : "MANAGED_USER",
    "properties" : {
        ...
        "defaultUserRoles" : [
            "internal/role/openidm-authorized"
        ]
    },
    ...
}
```

- The `userRoles` property in an authentication module maps to an attribute (or list of attributes) in a user entry that contains that user's authorization roles. This attribute is usually `authzRoles`, unless you have changed how user roles are stored.

  Any internal roles that are conditionally applied are also calculated and included in the user's `authzRoles` property at this point.

- If the authentication module includes a `groupRoleMapping`, `groupMembership`, or `groupComparison` property, IDM can assign additional roles to the user, depending on the user's group membership on an *external* system. For more information, refer to Use Groups to Control Access to IDM.

> **Note**
>
> The roles calculated in sequence are cumulative. Roles with temporal restrictions are not included in that list if the current time is outside of the time assigned to the role.

Dynamic role calculation

By default, IDM calculates a user's roles only on authentication. You can configure IDM to recalculate a user's roles dynamically, with each request, instead of only when the user reauthenticates. To enable this feature, set `enableDynamicRoles` to `true` in the `JWT_SESSION` session module in `authentication.json`:

To enable dynamic role calculation through the admin UI, click **Configure > Authentication > Session > Enable Dynamic Roles**.

Dynamic role calculation can be used independently of the *privileges* mechanism, but is required for privileges to work. For more information about privileges, refer to How Privileges Restrict Administrative Access.

Roles, authentication, and the security context

The Security Context (`context.security`), consists of a principal (defined by the `authenticationId` property) and an access control element (defined by the `authorization` property).

If authentication is successful, the authentication framework sets the principal. IDM stores that principal as the `authenticationId`.

The `authorization` property includes an `id`, an array of `roles`, and a `component`, that specifies the resource against which authorization is validated.

Authorization and roles

IDM provides role-based authorization that restricts direct HTTP access to REST interface URLs. This access control applies to direct HTTP calls only. Access for internal calls (for example, calls from scripts) is not affected by this mechanism.

When a user authenticates, they are given a set of default *roles*, as described in Authentication and Roles. The authorization configuration grants access rights to users, based on these roles acquired during authentication.

You can use internal and managed roles to restrict access, with the following caveats:

- Internal roles are not meant to be provisioned or synchronized with external systems.
- Internal roles cannot be given assignments.
- Event scripts (such as `onCreate`) cannot be attached to internal roles.
- The internal role schema is not configurable.

Authorization roles are referenced in a user's `authzRoles` property by default, and are assigned when the user authenticates.

By default, managed users are assigned the `openidm-authorized` role when they authenticate. The following request shows the authorization roles for user psmith when that user logs in to the server:

```
curl \
--header "X-OpenIDM-Username: psmith" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/info/login"
{
  "_id": "login",
  "authenticationId": "psmith",
  "authorization": {
    "userRolesProperty": "authzRoles",
    "component": "managed/user",
    "authLogin": false,
    "authenticationIdProperty": "username",
    "roles": [
      "internal/role/openidm-authorized"
    ],
    "ipAddress": "0:0:0:0:0:0:0:1",
    "authenticationId": "psmith",
    "protectedAttributeList": [
      "password"
    ],
    "id": "psmith",
    "moduleId": "MANAGED_USER",
    "queryId": "credential-query"
  }
}
```

The authorization implementation is configured in two files:

- `openidm/bin/defaults/script/router-authz.js`
- `project-dir/conf/access.json`

IDM calls the `router-authz.js` script for each request, through an `onRequest` hook defined in the `router.json` file. `router-authz.js` references your project's access configuration (`access.json`) to determine the allowed HTTP requests. If access is denied, according to the configuration defined in `access.json` , the `router-authz.js` script throws an exception, and IDM denies the request.

`router.json` also defines an `onResponse` script, `relationshipFilter` . This provides additional filtering to ensure that the user has the appropriate access to see the data of the related object. You can change this behavior by extending or updating `/bin/defaults/script/relationshipFilter.js` , or by removing the `onResponse` script if you don't want additional filtering on relationships. For more information about relationships, refer to Relationships between objects.

> **Note**
>
> You can configure delegated administration to grant access that bypasses this access control.

Modify and extend the router authorization script

The router authorization script (`router-authz.js` contains a number of functions that enforce access rules. For example, the following function controls whether users with a certain role can start a specified process:

```
function isAllowedToStartProcess() {
    var processDefinitionId = request.content._processDefinitionId;
    var key = request.content._key;
    return isProcessOnUsersList(function (process) {
        return (process._id === processDefinitionId) || (process.key === key);
    });
}
```

You can extend the default authorization mechanism by defining additional functions in `router-authz.js` and by creating new access control rules in `access.json` .

> **Important**
>
> Some authorization-related functions in `router-authz.js` should *not* be altered, because they affect the security of the server. Such functions are indicated in the comments in that file.

Configure access control in `access.json`

The `access.json` configuration includes a set of rules that govern access to specific endpoints. These rules are tested in the order in which they appear in the file. You can define more than one rule for the same endpoint. If one rule passes, the request is allowed. If all the rules fail, the request is denied.

The following rule (from a default `access.json` file) shows the access configuration structure:

```
{
    "pattern"   : "system/*",
    "roles"     : "internal/role/openidm-admin",
    "methods"   : "action",
    "actions"   : "test,testConfig,createconfiguration,liveSync,authenticate"
}
```

This rule specifies that users with the `openidm-admin` role can perform the listed actions on all `system` endpoints.

The parameters in each access rule are as follows:

`pattern`

The REST endpoint for which access is being controlled. `"*"` specifies access to all endpoints in that path. For example, `"managed/user/*"` specifies access to all managed user objects.

`roles`

A comma-separated list of the roles to which this access configuration applies.

The `roles` referenced here align with the object's security context (`security.authorization.roles`). The `authzRoles` relationship property of a managed user produces this security context value during authentication.

`methods`

A comma-separated list of the methods that can be performed with this access. Methods can include `create, read, update, delete, patch, action, query` . A value of `"*"` indicates that all methods are allowed. A value of `""` indicates that no methods are allowed.

`actions`

A comma-separated list of the allowed actions. The possible actions depend on the resource (URL) that is being exposed. Note that the `actions` in the default `access.json` file do not list all the supported actions on each resource.

A value of `"*"` indicates that all actions exposed for that resource are allowed. A value of `""` indicates that no actions are allowed.

`customAuthz`

An optional parameter that lets you define a custom function for additional authorization checks. Custom functions are defined in `router-authz.js` .

`excludePatterns`

An optional parameter that lets you specify endpoints to which access should not be granted.

Change the access configuration over REST

You can manage the access configuration at the endpoint `openidm/config/access` . To change an access rule, first get the current access configuration, amend it to change the access rule, then submit the updated configuration in a PUT request. This example restricts access to the `info` endpoint to users who have authenticated:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0"  \
--request GET \
"http://localhost:8080/openidm/config/access"
{
  "_id": "access",
  "configs": [
    {
      "pattern": "info/*",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "authentication",
      "roles": "*",
      "methods": "read,action",
      "actions": "login,logout"
    },
    {
      "pattern": "identityProviders",
      "roles": "*",
      "methods": "action",
      "actions": "getAuthRedirect,handlePostAuth,getLogoutUrl"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "normalizeProfile"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "config/ui/themeconfig",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "info/uiconfig",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "config/selfservice/kbaConfig",
      "roles": "*",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'passwordReset'])"
    },
    {
      "pattern": "config/ui/dashboard",
```

```
        "roles": "internal/role/openidm-authorized",
        "methods": "read",
        "actions": "*"
      },
      {
        "pattern": "info/features",
        "roles": "*",
        "methods": "query",
        "actions": "*"
      },
      {
        "pattern": "privilege",
        "roles": "*",
        "methods": "action",
        "actions": "listPrivileges"
      },
      {
        "pattern": "privilege/*",
        "roles": "*",
        "methods": "read",
        "actions": "*"
      },
      {
        "pattern": "selfservice/registration",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements",
        "customAuthz": "checkIfAnyFeatureEnabled('registration')"
      },
      {
        "pattern": "selfservice/socialUserClaim",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements",
        "customAuthz": "checkIfAnyFeatureEnabled('registration')"
      },
      {
        "pattern": "selfservice/reset",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements",
        "customAuthz": "checkIfAnyFeatureEnabled('passwordReset')"
      },
      {
        "pattern": "selfservice/username",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements",
        "customAuthz": "checkIfAnyFeatureEnabled('retrieveUsername')"
      },
      {
        "pattern": "selfservice/profile",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements"
      },
      {
        "pattern": "selfservice/termsAndConditions",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements"
      },
```

```
    {
      "pattern": "selfservice/kbaUpdate",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements"
    },
    {
      "pattern": "policy/*",
      "roles": "*",
      "methods": "action",
      "actions": "validateObject",
      "customAuthz": "context.current.name === 'selfservice'"
    },
    {
      "pattern": "policy/selfservice/registration",
      "roles": "*",
      "methods": "action,read",
      "actions": "validateObject",
      "customAuthz": "checkIfAnyFeatureEnabled('registration')"
    },
    {
      "pattern": "policy/selfservice/reset",
      "roles": "*",
      "methods": "action,read",
      "actions": "validateObject",
      "customAuthz": "checkIfAnyFeatureEnabled('passwordReset')"
    },
    {
      "pattern": "selfservice/kba",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled('kba')"
    },
    {
      "pattern": "managed/user",
      "roles": "internal/role/openidm-reg",
      "methods": "create",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled('registration') && isSelfServiceRequest() &&
onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "managed/user",
      "roles": "*",
      "methods": "query",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'retrieveUsername', 'passwordReset']) &&
isSelfServiceRequest()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "*",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['retrieveUsername', 'passwordReset']) && isSelfServiceRequest()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "*",
      "methods": "patch,action",
      "actions": "patch",
```

```
      "customAuthz": "(checkIfAnyFeatureEnabled(['registration', 'passwordReset']) ||
checkIfProgressiveProfileIsEnabled()) && isSelfServiceRequest() && onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "external/email",
      "roles": "*",
      "methods": "action",
      "actions": "send",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'retrieveUsername', 'passwordReset']) &&
isSelfServiceRequest()"
    },
    {
      "pattern": "schema/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "consent",
      "roles": "internal/role/openidm-authorized",
      "methods": "action,query",
      "actions": "*"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "excludePatterns": "repo,repo/*"
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "create,read,update,delete,patch,query",
      "actions": ""
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "script",
      "actions": "*"
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "action",
      "actions": "test,testConfig,createconfiguration,liveSync,authenticate"
    },
    {
      "pattern": "repo",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "customAuthz": "disallowCommandAction()"
    },
    {
      "pattern": "repo/*",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "customAuthz": "disallowCommandAction()"
    },
```

```
    {
      "pattern": "repo/link",
      "roles": "internal/role/openidm-admin",
      "methods": "action",
      "actions": "command",
      "customAuthz": "request.additionalParameters.commandId === 'delete-mapping-links'"
    },
    {
      "pattern": "managed/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "create,read,query,patch"
    },
    {
      "pattern": "internal/role/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read,query"
    },
    {
      "pattern": "profile/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "create,read,action,update",
      "actions": "*"
    },
    {
      "pattern": "policy/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read,action",
      "actions": "*"
    },
    {
      "pattern": "schema/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "consent",
      "roles": "internal/role/platform-provisioning",
      "methods": "action,query",
      "actions": "*"
    },
    {
      "pattern": "selfservice/kba",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "selfservice/terms",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "external/email",
      "roles": "internal/role/platform-provisioning",
      "methods": "action",
      "actions": "sendTemplate"
    },
    {
```

```
      "pattern": "policy/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,action",
      "actions": "*"
    },
    {
      "pattern": "config/ui/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "authentication",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "reauthenticate"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,action,delete",
      "actions": "bind,unbind",
      "customAuthz": "ownDataOnly()"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-authorized",
      "methods": "update,patch,action",
      "actions": "patch",
      "customAuthz": "ownDataOnly() && onlyEditableManagedObjectProperties('user', []) &&
reauthIfProtectedAttributeChange()"
    },
    {
      "pattern": "selfservice/user/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "(request.resourcePath === 'selfservice/user/' + context.security.authorization.id) &&
onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "endpoint/getprocessesforuser",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "endpoint/gettasksview",
      "roles": "internal/role/openidm-authorized",
      "methods": "query",
      "actions": "*"
    },
    {
      "pattern": "workflow/taskinstance/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "complete",
      "customAuthz": "isMyTask()"
    },
    {
      "pattern": "workflow/taskinstance/*",
      "roles": "internal/role/openidm-authorized",
```

```
      "methods": "read,update",
      "actions": "*",
      "customAuthz": "canUpdateTask()"
    },
    {
      "pattern": "workflow/processinstance",
      "roles": "internal/role/openidm-authorized",
      "methods": "create",
      "actions": "*",
      "customAuthz": "isAllowedToStartProcess()"
    },
    {
      "pattern": "workflow/processdefinition/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "*",
      "actions": "read",
      "customAuthz": "isOneOfMyWorkflows()"
    },
    {
      "pattern": "managed/user",
      "roles": "internal/role/openidm-cert",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "isQueryOneOf({'managed/user': ['for-userName']}) && restrictPatchToFields(['password'])"
    },
    {
      "pattern": "internal/usermeta/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "ownRelationship()"
    },
    {
      "pattern": "internal/notification/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,delete",
      "actions": "*",
      "customAuthz": "ownRelationship()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,query",
      "actions": "*",
      "customAuthz": "ownRelationshipCollection(['idps','_meta','_notifications'])"
    },
    {
      "pattern": "notification",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "deleteNotificationsForTarget",
      "customAuthz": "request.additionalParameters.target === (context.security.authorization.component + '/' +
context.security.authorization.id)"
    },
    {
      "pattern": "managed/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
```

```
        "actions": "*",
        "customAuthz": "ownIDP()"
      }
    ]
  }
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-type: application/json" \
--header "Accept-API-Version: resource=1.0"  \
--request PUT \
--data '{
  "_id": "access",
  "configs": [
    {
      "pattern": "info/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "authentication",
      "roles": "*",
      "methods": "read,action",
      "actions": "login,logout"
    },
    {
      "pattern": "identityProviders",
      "roles": "*",
      "methods": "action",
      "actions": "getAuthRedirect,handlePostAuth,getLogoutUrl"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "normalizeProfile"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "config/ui/themeconfig",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "info/uiconfig",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "config/selfservice/kbaConfig",
      "roles": "*",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'passwordReset'])"
    },
    {
      "pattern": "config/ui/dashboard",
```

```
        "roles": "internal/role/openidm-authorized",
        "methods": "read",
        "actions": "*"
      },
      {
        "pattern": "info/features",
        "roles": "*",
        "methods": "query",
        "actions": "*"
      },
      {
        "pattern": "privilege",
        "roles": "*",
        "methods": "action",
        "actions": "listPrivileges"
      },
      {
        "pattern": "privilege/*",
        "roles": "*",
        "methods": "read",
        "actions": "*"
      },
      {
        "pattern": "selfservice/registration",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements",
        "customAuthz": "checkIfAnyFeatureEnabled('registration')"
      },
      {
        "pattern": "selfservice/socialUserClaim",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements",
        "customAuthz": "checkIfAnyFeatureEnabled('registration')"
      },
      {
        "pattern": "selfservice/reset",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements",
        "customAuthz": "checkIfAnyFeatureEnabled('passwordReset')"
      },
      {
        "pattern": "selfservice/username",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements",
        "customAuthz": "checkIfAnyFeatureEnabled('retrieveUsername')"
      },
      {
        "pattern": "selfservice/profile",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements"
      },
      {
        "pattern": "selfservice/termsAndConditions",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements"
      },
```

```
    {
      "pattern": "selfservice/kbaUpdate",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements"
    },
    {
      "pattern": "policy/*",
      "roles": "*",
      "methods": "action",
      "actions": "validateObject",
      "customAuthz": "context.current.name === 'selfservice'"
    },
    {
      "pattern": "policy/selfservice/registration",
      "roles": "*",
      "methods": "action,read",
      "actions": "validateObject",
      "customAuthz": "checkIfAnyFeatureEnabled('registration')"
    },
    {
      "pattern": "policy/selfservice/reset",
      "roles": "*",
      "methods": "action,read",
      "actions": "validateObject",
      "customAuthz": "checkIfAnyFeatureEnabled('passwordReset')"
    },
    {
      "pattern": "selfservice/kba",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled('kba')"
    },
    {
      "pattern": "managed/user",
      "roles": "internal/role/openidm-reg",
      "methods": "create",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled('registration') && isSelfServiceRequest() &&
onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "managed/user",
      "roles": "*",
      "methods": "query",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'retrieveUsername', 'passwordReset']) &&
isSelfServiceRequest()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "*",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['retrieveUsername', 'passwordReset']) && isSelfServiceRequest()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "*",
      "methods": "patch,action",
      "actions": "patch",
```

```
      "customAuthz": "(checkIfAnyFeatureEnabled(['registration', 'passwordReset']) ||
checkIfProgressiveProfileIsEnabled()) && isSelfServiceRequest() && onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "external/email",
      "roles": "*",
      "methods": "action",
      "actions": "send",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'retrieveUsername', 'passwordReset']) &&
isSelfServiceRequest()"
    },
    {
      "pattern": "schema/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "consent",
      "roles": "internal/role/openidm-authorized",
      "methods": "action,query",
      "actions": "*"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "excludePatterns": "repo,repo/*"
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "create,read,update,delete,patch,query",
      "actions": ""
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "script",
      "actions": "*"
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "action",
      "actions": "test,testConfig,createconfiguration,liveSync,authenticate"
    },
    {
      "pattern": "repo",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "customAuthz": "disallowCommandAction()"
    },
    {
      "pattern": "repo/*",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "customAuthz": "disallowCommandAction()"
    },
```

```
    {
      "pattern": "repo/link",
      "roles": "internal/role/openidm-admin",
      "methods": "action",
      "actions": "command",
      "customAuthz": "request.additionalParameters.commandId === 'delete-mapping-links'"
    },
    {
      "pattern": "managed/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "create,read,query,patch"
    },
    {
      "pattern": "internal/role/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read,query"
    },
    {
      "pattern": "profile/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "create,read,action,update",
      "actions": "*"
    },
    {
      "pattern": "policy/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read,action",
      "actions": "*"
    },
    {
      "pattern": "schema/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "consent",
      "roles": "internal/role/platform-provisioning",
      "methods": "action,query",
      "actions": "*"
    },
    {
      "pattern": "selfservice/kba",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "selfservice/terms",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "external/email",
      "roles": "internal/role/platform-provisioning",
      "methods": "action",
      "actions": "sendTemplate"
    },
    {
```

```
      "pattern": "policy/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,action",
      "actions": "*"
    },
    {
      "pattern": "config/ui/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "authentication",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "reauthenticate"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,action,delete",
      "actions": "bind,unbind",
      "customAuthz": "ownDataOnly()"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-authorized",
      "methods": "update,patch,action",
      "actions": "patch",
      "customAuthz": "ownDataOnly() && onlyEditableManagedObjectProperties('user', []) &&
reauthIfProtectedAttributeChange()"
    },
    {
      "pattern": "selfservice/user/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "(request.resourcePath === 'selfservice/user/' + context.security.authorization.id) &&
onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "endpoint/getprocessesforuser",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "endpoint/gettasksview",
      "roles": "internal/role/openidm-authorized",
      "methods": "query",
      "actions": "*"
    },
    {
      "pattern": "workflow/taskinstance/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "complete",
      "customAuthz": "isMyTask()"
    },
    {
      "pattern": "workflow/taskinstance/*",
      "roles": "internal/role/openidm-authorized",
```

```
      "methods": "read,update",
      "actions": "*",
      "customAuthz": "canUpdateTask()"
    },
    {
      "pattern": "workflow/processinstance",
      "roles": "internal/role/openidm-authorized",
      "methods": "create",
      "actions": "*",
      "customAuthz": "isAllowedToStartProcess()"
    },
    {
      "pattern": "workflow/processdefinition/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "*",
      "actions": "read",
      "customAuthz": "isOneOfMyWorkflows()"
    },
    {
      "pattern": "managed/user",
      "roles": "internal/role/openidm-cert",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "isQueryOneOf({'managed/user': ['for-userName']}) && restrictPatchToFields(['password'])"
    },
    {
      "pattern": "internal/usermeta/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "ownRelationship()"
    },
    {
      "pattern": "internal/notification/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,delete",
      "actions": "*",
      "customAuthz": "ownRelationship()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,query",
      "actions": "*",
      "customAuthz": "ownRelationshipCollection(['idps','_meta','_notifications'])"
    },
    {
      "pattern": "notification",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "deleteNotificationsForTarget",
      "customAuthz": "request.additionalParameters.target === (context.security.authorization.component + '/' +
context.security.authorization.id)"
    },
    {
      "pattern": "managed/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "ownIDP()"
    }
  ]
```

```
}' \
"http://localhost:8080/openidm/config/access"
{
  "_id": "access",
  "configs": [
    {
      "pattern": "info/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "authentication",
      "roles": "*",
      "methods": "read,action",
      "actions": "login,logout"
    },
    {
      "pattern": "identityProviders",
      "roles": "*",
      "methods": "action",
      "actions": "getAuthRedirect,handlePostAuth,getLogoutUrl"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "normalizeProfile"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "config/ui/themeconfig",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "info/uiconfig",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "config/selfservice/kbaConfig",
      "roles": "*",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'passwordReset'])"
    },
    {
      "pattern": "config/ui/dashboard",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "info/features",
```

```
        "roles": "*",
        "methods": "query",
        "actions": "*"
      },
      {
        "pattern": "privilege",
        "roles": "*",
        "methods": "action",
        "actions": "listPrivileges"
      },
      {
        "pattern": "privilege/*",
        "roles": "*",
        "methods": "read",
        "actions": "*"
      },
      {
        "pattern": "selfservice/registration",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements",
        "customAuthz": "checkIfAnyFeatureEnabled('registration')"
      },
      {
        "pattern": "selfservice/socialUserClaim",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements",
        "customAuthz": "checkIfAnyFeatureEnabled('registration')"
      },
      {
        "pattern": "selfservice/reset",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements",
        "customAuthz": "checkIfAnyFeatureEnabled('passwordReset')"
      },
      {
        "pattern": "selfservice/username",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements",
        "customAuthz": "checkIfAnyFeatureEnabled('retrieveUsername')"
      },
      {
        "pattern": "selfservice/profile",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements"
      },
      {
        "pattern": "selfservice/termsAndConditions",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements"
      },
      {
        "pattern": "selfservice/kbaUpdate",
        "roles": "*",
        "methods": "read,action",
        "actions": "submitRequirements"
      },
```

```
    {
      "pattern": "policy/*",
      "roles": "*",
      "methods": "action",
      "actions": "validateObject",
      "customAuthz": "context.current.name === 'selfservice'"
    },
    {
      "pattern": "policy/selfservice/registration",
      "roles": "*",
      "methods": "action,read",
      "actions": "validateObject",
      "customAuthz": "checkIfAnyFeatureEnabled('registration')"
    },
    {
      "pattern": "policy/selfservice/reset",
      "roles": "*",
      "methods": "action,read",
      "actions": "validateObject",
      "customAuthz": "checkIfAnyFeatureEnabled('passwordReset')"
    },
    {
      "pattern": "selfservice/kba",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled('kba')"
    },
    {
      "pattern": "managed/user",
      "roles": "internal/role/openidm-reg",
      "methods": "create",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled('registration') && isSelfServiceRequest() &&
onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "managed/user",
      "roles": "*",
      "methods": "query",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'retrieveUsername', 'passwordReset']) &&
isSelfServiceRequest()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "*",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['retrieveUsername', 'passwordReset']) && isSelfServiceRequest()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "*",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "(checkIfAnyFeatureEnabled(['registration', 'passwordReset']) ||
checkIfProgressiveProfileIsEnabled()) && isSelfServiceRequest() && onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "external/email",
      "roles": "*",
```

```
      "methods": "action",
      "actions": "send",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'retrieveUsername', 'passwordReset']) &&
isSelfServiceRequest()"
    },
    {
      "pattern": "schema/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "consent",
      "roles": "internal/role/openidm-authorized",
      "methods": "action,query",
      "actions": "*"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "excludePatterns": "repo,repo/*"
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "create,read,update,delete,patch,query",
      "actions": ""
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "script",
      "actions": "*"
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "action",
      "actions": "test,testConfig,createconfiguration,liveSync,authenticate"
    },
    {
      "pattern": "repo",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "customAuthz": "disallowCommandAction()"
    },
    {
      "pattern": "repo/*",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "customAuthz": "disallowCommandAction()"
    },
    {
      "pattern": "repo/link",
      "roles": "internal/role/openidm-admin",
      "methods": "action",
      "actions": "command",
      "customAuthz": "request.additionalParameters.commandId === 'delete-mapping-links'"
```

```
    },
    {
      "pattern": "managed/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "create,read,query,patch"
    },
    {
      "pattern": "internal/role/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read,query"
    },
    {
      "pattern": "profile/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "create,read,action,update",
      "actions": "*"
    },
    {
      "pattern": "policy/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read,action",
      "actions": "*"
    },
    {
      "pattern": "schema/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "consent",
      "roles": "internal/role/platform-provisioning",
      "methods": "action,query",
      "actions": "*"
    },
    {
      "pattern": "selfservice/kba",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "selfservice/terms",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "external/email",
      "roles": "internal/role/platform-provisioning",
      "methods": "action",
      "actions": "sendTemplate"
    },
    {
      "pattern": "policy/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,action",
      "actions": "*"
    },
    {
```

```
      "pattern": "config/ui/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "authentication",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "reauthenticate"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,action,delete",
      "actions": "bind,unbind",
      "customAuthz": "ownDataOnly()"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-authorized",
      "methods": "update,patch,action",
      "actions": "patch",
      "customAuthz": "ownDataOnly() && onlyEditableManagedObjectProperties('user', []) &&
reauthIfProtectedAttributeChange()"
    },
    {
      "pattern": "selfservice/user/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "(request.resourcePath === 'selfservice/user/' + context.security.authorization.id) &&
onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "endpoint/getprocessesforuser",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "endpoint/gettasksview",
      "roles": "internal/role/openidm-authorized",
      "methods": "query",
      "actions": "*"
    },
    {
      "pattern": "workflow/taskinstance/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "complete",
      "customAuthz": "isMyTask()"
    },
    {
      "pattern": "workflow/taskinstance/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,update",
      "actions": "*",
      "customAuthz": "canUpdateTask()"
    },
    {
      "pattern": "workflow/processinstance",
```

```
      "roles": "internal/role/openidm-authorized",
      "methods": "create",
      "actions": "*",
      "customAuthz": "isAllowedToStartProcess()"
    },
    {
      "pattern": "workflow/processdefinition/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "*",
      "actions": "read",
      "customAuthz": "isOneOfMyWorkflows()"
    },
    {
      "pattern": "managed/user",
      "roles": "internal/role/openidm-cert",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "isQueryOneOf({'managed/user': ['for-userName']}) && restrictPatchToFields(['password'])"
    },
    {
      "pattern": "internal/usermeta/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "ownRelationship()"
    },
    {
      "pattern": "internal/notification/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,delete",
      "actions": "*",
      "customAuthz": "ownRelationship()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,query",
      "actions": "*",
      "customAuthz": "ownRelationshipCollection(['idps','_meta','_notifications'])"
    },
    {
      "pattern": "notification",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "deleteNotificationsForTarget",
      "customAuthz": "request.additionalParameters.target === (context.security.authorization.component + '/' +
context.security.authorization.id)"
    },
    {
      "pattern": "managed/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "ownIDP()"
    }
  ]
}
```

Grant internal authorization roles manually

Apart from the default roles that users get when they authenticate, you can grant internal authorization roles manually, over REST or using the admin UI. This mechanism works in the same way as the granting of managed roles. For information about granting managed roles, refer to Grant Roles to a User. To grant an internal role manually through the admin UI:

1. From the navigation bar, click **Manage > User**, and click a user.
2. From the **Authorization Roles** tab, click **Add Authorization Roles**.
3. Select **Internal Role** as the **Type**, click in the **Authorization Roles** field to select from the list of defined **Internal Roles**, and click **Add**.

To manually grant an internal role over REST, add a reference to the internal role to the user's `authzRoles` property. The following command adds the `openidm-admin` role to user bjensen (with ID `9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb`):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/authzRoles/-",
    "value": {"_ref" : "internal/role/openidm-admin"}
  }
]' \
"https://localhost:8443/openidm/managed/user/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb"
{
  "_id": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
  "_rev": "0000000050c62938",
  "mail": "bjensen@example.com",
  "givenName": "Barbara",
  "sn": "Jensen",
  "description": "Created By CSV",
  "userName": "bjensen",
  "telephoneNumber": "1234567",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

You can also grant internal roles dynamically using conditional role grants.

Note

Because internal roles are not managed objects, you cannot manipulate them in the same way as managed roles. Therefore, you cannot add a user to an internal role, as you would to a managed role.
To add users directly to an internal role, add the users as values of the role's `authzMembers` property. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request POST \
--data '{"_ref":"managed/user/bjensen"}' \
"https://localhost:8443/openidm/internal/role/3042798d-37fd-49aa-bae3-52598d2c8dc4/authzMembers?
_action=create"
```

Secure access to workflows

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

A sample workflow integration with the End User UI is provided in `openidm/samples/provisioning-with-workflow` , and documented in Provision users with workflow. Follow the steps in that sample for an understanding of how the workflow integration works.

General access to workflow-related endpoints is based on the access rules defined in the `conf/access.json` file. The configuration defined in `conf/process-access.json` specifies who can invoke workflows. By default, all users with the role `openidm-authorized` or `openidm-admin` can invoke any available workflow. The default `process-access.json` file is as follows:

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

`property`

Specifies the property used to identify the process definition. By default, process definitions are identified by their `_id` .

`matches`

A regular expression match is performed on the process definitions, according to the specified property. The default (`"matches" : ".*"`) implies that all process definition IDs match.

`requiresRole`

Specifies the authorization role that is required for users to have access to the matched process definition IDs. In the default file, users with the role `openidm-authorized` or `openidm-admin` have access.

To extend the process action definition file, identify the processes to which users should have access, and specify the qualifying authorization roles. For example, if you want to allow access to users with a role of `ldap`, add the following code block to the `process-access.json` file:

```
{
    "propertiesCheck" : {
        "property" : "_id",
        "matches" : ".*",
        "requiresRole" : "ldap"
    }
}
```

To configure multiple roles with access to the same workflow process, simply add additional `propertiesCheck` objects. The following example grants access to users with a role of doctor and nurse to the same workflows:

```
{
     "propertiesCheck" : {
          "property" : "_id",
          "matches" : ".*",
          "requiresRole" : "doctor"
     }
},
{
     "propertiesCheck" : {
          "property" : "_id",
          "matches" : ".*",
          "requiresRole" : "nurse"
     }
}
```

Secure RCS access

You can secure the `openicf` WebSocket endpoint used for communication between IDM and RCS client mode. Specifying the authorization roles allowed to connect to the `openicf` servlet for each RCS instance ensures that RCS access to IDM is protected can only be accessed by trusted RCS clients. The `openicf` servlet in your access configuration (`access.json`) lets you manage authorization for the `openicf` endpoint.

1. In your authentication configuration (`conf/authentication.json`), add a specific role to the `roles` array of the RCS static user mapping. This role authorizes access to the `openicf` endpoint for the specified RCS.

Example

```
{
    "subject": "rcsclient",
    "localUser": "internal/user/idm-provisioning",
    "roles": [
        "internal/role/myrcsserver-authorized" (1)
    ],
    "executeAugmentationScript": false
}
```

1 In this example, the role `internal/role/myrcsserver-authorized` is added to the authenticated user's security context.

2. Add a new rule in your access configuration (`conf/access.json`) to define authorization for the `openicf` endpoint.

Example

```
{
    "servlet": "openicf", (2)
    "pattern": "myrcsserver", (3)
    "roles": "internal/role/myrcsserver-authorized", (1)
    "methods": "read"
}
```

This rule ensures that:

1 only users possessing the `internal/role/myrcsserver-authorized` role can access the
2 `openicf` WebSocket endpoint
3 for the RCS named `myrcsserver`.

Administrative users

The default IDM administrative user is `openidm-admin` . In a production environment, you might want to replace this user with a managed or internal user with the same roles, specifically the `openidm-admin` and `openidm-authorized` roles.

You can create either an internal or managed user with the same roles as the default `openidm-admin` user. To add these roles to an existing managed user, refer to Grant Internal Authorization Roles Manually. The following procedure creates a new administrative internal user (`admin`):

1. Create an internal user:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request PUT \
--data '{
  "password": "Passw0rd"
}' \
"https://localhost:8443/openidm/internal/user/admin"
{
  "_id": "admin",
  "_rev": "00000000210f6746"
}
```

2. Add a `STATIC_USER` authentication module to the authentication configuration:

   Edit the `conf/authentication.json` file, and add the following:

```
{
  "name" : "STATIC_USER",
  "properties" : {
    "queryOnResource" : "internal/user",
    "username" : "admin",
    "password" : "Passw0rd",
    "defaultUserRoles" : [
      "internal/role/openidm-authorized",
      "internal/role/openidm-admin"
    ]
  },
  "enabled" : true
}
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/serverAuthContext/authModules/-",
    "value": {
      "name" : "STATIC_USER",
      "properties" : {
        "queryOnResource" : "internal/user",
        "username" : "admin",
        "password" : "Passw0rd",
        "defaultUserRoles" : [
          "internal/role/openidm-authorized",
          "internal/role/openidm-admin"
        ]
      },
      "enabled" : true
    }
  }
]' \
"https://localhost:8443/openidm/config/authentication"
{
  "_id": "authentication",
  "serverAuthContext": {
    ...
    "authModules": [
      ...
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "admin",
          "password": "{encrypted password}",
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      ...
    ]
  }
}
```

3. To verify the changes, perform a REST call or log in to the admin UI as the new admin user. For example, query the list of internal users:

```
curl \
--header "X-OpenIDM-Username: admin" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/internal/user?_queryFilter=true"
{
  "result": [
    {
      "_id": "admin",
      "_rev": "00000000f8e1665a"
    }
  ],
  ...
}
```

4. *After* you have verified the new admin user, you can delete or disable the `openidm-admin` user:

    1. Delete the `openidm-admin` object:

    ```
    curl \
    --header "X-OpenIDM-Username: admin" \
    --header "X-OpenIDM-Password: Passw0rd" \
    --header "Accept-API-Version: resource=1.0" \
    --cacert ca-cert.pem \
    --request DELETE \
    "https://localhost:8443/openidm/internal/user/openidm-admin"
    {
      "_id": "openidm-admin",
      "_rev": "00000000210f6746"
    }
    ```

    2. Delete the authentication module for `"username" : "openidm-admin"`:

    Edit the `conf/authentication.json` file, and delete:

    ```
    {
      "name" : "STATIC_USER",
      "properties" : {
        "queryOnResource" : "internal/user",
        "username" : "openidm-admin",
        "password" : "&{openidm.admin.password}",
        "defaultUserRoles" : [
          "internal/role/openidm-authorized",
          "internal/role/openidm-admin"
        ]
      },
      "enabled" : true
    }
    ```

        1. Get the current authentication configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/config/authentication"
{
  "_id": "authentication",
  "serverAuthContext": {
    ...
    "authModules": [
      ...
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "openidm-admin",
          "password": "&{openidm.admin.password}",
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      ...
    ]
  }
}
```

2. Remove the authentication module for `"username" : "openidm-admin"`, and replace the authentication configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request PUT \
--data '{
  "_id": "authentication",
  "serverAuthContext": {
    "sessionModule": {
      "name": "JWT_SESSION",
      "properties": {
        "maxTokenLifeMinutes": 120,
        "tokenIdleTimeMinutes": 30,
        "sessionOnly": true,
        "isHttpOnly": true,
        "enableDynamicRoles": false
      }
    },
    "authModules": [
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "anonymous",
          "password": {
            "$crypto": {
              "type": "x-simple-encryption",
              "value": {
                "cipher": "AES/CBC/PKCS5Padding",
                "stableId": "openidm-sym-default",
                "salt": "xBlTp67ze4Ca5LTocXOpoA==",
                "data": "mdibV6UabU2M+M5MK7bjFQ==",
                "keySize": 16,
                "purpose": "idm.config.encryption",
                "iv": "36D2+FumKbaUsndNQ+/+5w==",
                "mac": "ZM8GMnh0n80QwtSH6QsNmA=="
              }
            }
          },
          "defaultUserRoles": [
            "internal/role/openidm-reg"
          ]
        },
        "enabled": true
      },
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "admin",
          "password": "{encrypted password}",
          "defaultUserRoles": [
```

```
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      {
        "name": "MANAGED_USER",
        "properties": {
          "augmentSecurityContext": {
            "type": "text/javascript",
            "source": "require('auth/customAuthz').setProtectedAttributes(security)"
          },
          "queryId": "credential-query",
          "queryOnResource": "managed/user",
          "propertyMapping": {
            "authenticationId": "username",
            "userCredential": "password",
            "userRoles": "authzRoles"
          },
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ]
        },
        "enabled": true
      },
      {
        "name": "SOCIAL_PROVIDERS",
        "properties": {
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ],
          "augmentSecurityContext": {
            "type": "text/javascript",
            "globals": {},
            "file": "auth/populateAsManagedUserFromRelationship.js"
          },
          "propertyMapping": {
            "userRoles": "authzRoles"
          }
        },
        "enabled": true
      }
    ]
  }
}' \
"https://localhost:8443/openidm/config/authentication"
```

3. Prevent the `openidm-admin` user from being recreated on startup.

   Delete the following lines from the `internal/user` array in `conf/repo.init.json`:

```
{
    "id" : "openidm-admin",
    "password" : "&{openidm.admin.password}"
}
```

Change the `enabled` state of the authentication module for `"username" : "openidm-admin"`:

Edit the `conf/authentication.json` file:

```
{
  "name" : "STATIC_USER",
  "properties" : {
    "queryOnResource" : "internal/user",
    "username" : "openidm-admin",
    "password" : "&{openidm.admin.password}",
    "defaultUserRoles" : [
      "internal/role/openidm-authorized",
      "internal/role/openidm-admin"
    ]
  },
  "enabled" : false
}
```

1. Get the current authentication configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/config/authentication"
{
  "_id": "authentication",
  "serverAuthContext": {
    ...
    "authModules": [
      ...
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "openidm-admin",
          "password": "&{openidm.admin.password}",
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      ...
    ]
  }
}
```

2. Change the enabled state of the authentication module for `"username" : "openidm-admin"`, and replace the authentication configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request PUT \
--data '{
  "_id": "authentication",
  "serverAuthContext": {
    "sessionModule": {
      "name": "JWT_SESSION",
      "properties": {
        "maxTokenLifeMinutes": 120,
        "tokenIdleTimeMinutes": 30,
        "sessionOnly": true,
        "isHttpOnly": true,
        "enableDynamicRoles": false
      }
    },
    "authModules": [
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "anonymous",
          "password": {
            "$crypto": {
              "type": "x-simple-encryption",
              "value": {
                "cipher": "AES/CBC/PKCS5Padding",
                "stableId": "openidm-sym-default",
                "salt": "xBlTp67ze4Ca5LTocXOpoA==",
                "data": "mdibV6UabU2M+M5MK7bjFQ==",
                "keySize": 16,
                "purpose": "idm.config.encryption",
                "iv": "36D2+FumKbaUsndNQ+/+5w==",
                "mac": "ZM8GMnh0n80QwtSH6QsNmA=="
              }
            }
          },
          "defaultUserRoles": [
            "internal/role/openidm-reg"
          ]
        },
        "enabled": true
      },
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "openidm-admin",
          "password": "&{openidm.admin.password}",
          "defaultUserRoles": [
```

```
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": false
      },
      {
        "name": "MANAGED_USER",
        "properties": {
          "augmentSecurityContext": {
            "type": "text/javascript",
            "source": "require('auth/customAuthz').setProtectedAttributes(security)"
          },
          "queryId": "credential-query",
          "queryOnResource": "managed/user",
          "propertyMapping": {
            "authenticationId": "username",
            "userCredential": "password",
            "userRoles": "authzRoles"
          },
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ]
        },
        "enabled": true
      },
      {
        "name": "SOCIAL_PROVIDERS",
        "properties": {
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ],
          "augmentSecurityContext": {
            "type": "text/javascript",
            "globals": {},
            "file": "auth/populateAsManagedUserFromRelationship.js"
          },
          "propertyMapping": {
            "userRoles": "authzRoles"
          }
        },
        "enabled": true
      }
    ]
  }
}' \
"https://localhost:8443/openidm/config/authentication"
```

Delegated administration

Delegated administration lets you give fine-grained administrative access to specific users, based on a *privilege* mechanism.

How privileges restrict administrative access

Privileges enable you to grant administrative access to specific endpoints and objects, without needing to grant full administrative access to the server. For example, you might want to allow users with a help desk or support role to update the information of another user, without allowing them to delete user accounts or change the IDM system configuration.

You can use privileges to delegate specific administrative capabilities to non-administrative users, without exposing the admin UI to those users. If a user has been granted a privilege that allows them to access a list of users and user information, for example, they can access this list directly through the End User UI.

> **Note**
>
> A delegated administrator does not have access to the same methods over REST as a regular administrator. IDM does not allow delegated administrator requests such as POST or DELETE. To add or remove relationships, use PATCH. For examples, refer to Managed roles.

The privilege mechanism requires dynamic role calculation, which is disabled by default. To enable it, set the `enableDynamicRoles` property to `true` in your `conf/authentication.json` file, or select **Configure > Authentication > Session > Enable Dynamic Roles** in the admin UI. For more information about dynamic role calculation, refer to Dynamic Role Calculation.

For more information on managing privileges over REST, refer to Privileges.

Determine access privileges

IDM determines what access a user has as follows:

1. IDM checks the `onRequest` script specified in `router.json` . By default, this script calls `router-authz.js` .
2. If access requirements are not satisfied, IDM then checks for any privileges associated with the user's roles.

`onResponse` and `onFailure` scripts are supported when using privileges. `onFailure` scripts are called only if both the `onRequest` script *and* the privilege filter fail. `onRequest` , `onResponse` , and `onFailure` scripts are not required for the privilege mechanism.

Create privileges

Privileges are assigned to internal roles. A privilege specifies the following information:

- The service path where users with that internal role have access.
- The methods and actions allowed on that service path.
- The specific attributes of the objects at that service path where access is allowed.

You can use a query filter within a privilege so that the privilege applies to a subset of managed objects only.

The `privileges` property is an array and can contain multiple privileges. Each privilege can contain:

accessFlags

A list of attributes within a managed object that you want to give access to. Each attribute has two fields:

Field	Description
`attribute`	The name of the property you are granting access to.
`readOnly` (boolean)	Determines what level of access is allowed.

- Attributes marked as `"readOnly": true` can be viewed, but not edited.
- Attributes marked as `"readOnly": false` can be viewed and edited.
- Attributes that aren't listed in the `accessFlags` array cannot be viewed or edited.

> **Note**
>
> - Privileges aren't automatically aware of changes to the managed object schema. If new properties are added, removed, or made mandatory, you must update any existing privileges to account for these changes. When a new property is added, it has a default permission level of `NONE` in existing privileges, including when the privilege is set to access all attributes.
> - Identity Management applies policy validation when creating or updating a privilege to ensure that all required properties are writable when the `CREATE` permission is assigned. This validation doesn't run when schema changes are made, so you must verify that any existing privileges adhere to defined policies.

actions

A list of the specific actions allowed if the `ACTION` permission has been specified.

> **Tip**
>
> Allowed actions must be explicitly listed.

description (optional)

A description of the privilege.

filter (optional)

This property lets you apply a static or dynamic query filter to the privilege, which can be used to limit the scope of what the privilege allows the user to access.

Static filter example

To allow a delegated administrator to access information only about users for the `stateProvince` of Washington, include a static filter, such as:

```
filter : "stateProvince eq \"Washington\""
```

Dynamic filter example

Dynamic filters insert values from the authenticated resource. To allow a delegated administrator to access information only about users in their own `stateProvince`, include a dynamic filter by wrapping the parameter in curly braces:

```
filter : "stateProvince eq \"{{stateProvince}}\""
```

Users with query filter privileges can't edit the properties specified in the filter in ways that would cause the privilege to lose access to the object. For example, if a user with either of the preceding example privileges attempted to edit another user's `stateProvince` field to a value not matching the query filter, the request would return a `403 Forbidden` error.

> **Note**
>
> Fields must be *searchable* by Identity Management to be used in a privilege filter. Ensure that the field you are filtering has `"searchable" : true` set in the `repo.jdbc.json` file.
> Privilege filters are an additional layer of filter to any other query filters you create. Any output must satisfy all filters to be included.

name

The name of the privilege.

path

The path to the service that you want to allow members of this privilege to access. For example, `managed/user`.

permissions

A list of permissions this privilege allows for the given path. The following permissions are available:

`VIEW`	Allows reading and querying the path, such as viewing and querying managed users.
`CREATE`	Allows creation at the path, such as creating new managed users.
`UPDATE`	Allows updating or patching existing information, such as editing managed user details.
`DELETE`	Allows deletion, such as deleting users from `managed/user`.
`ACTION`	Allows users to perform actions at the given path, such as custom scripted actions.

> **Important**
>
> If you use an `ACTION`, there can be no filters on the privilege.

Adding privileges using the admin UI

1. From the navigation bar, click **Manage > Role**.
2. On the **Roles** page, click the **Internal** tab, and then click an existing role or create a new role.
3. On the **Role Name** page, click the **Privileges** tab.

   Identity Management displays the current privileges for the role.
4. To add privileges, click **Add Privileges**.
   - In the **Add a privilege** window, enter information, as necessary, and click **Add**.

Adding privileges using REST

The following example creates a new `support` role with privileges that let members view, create, and update information about users, but not delete users:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request PUT \
--data '{
  "name": "support",
  "description": "Support Role",
  "privileges": [ {
    "name": "support",
    "description": "Support access to user information.",
    "path": "managed/user",
    "permissions": [
      "VIEW", "UPDATE", "CREATE"
    ],
    "actions": [],
    "filter": null,
    "accessFlags": [
      {
        "attribute" : "userName",
        "readOnly" : false
      },
      {
        "attribute" : "mail",
        "readOnly" : false
      },
      {
        "attribute" : "givenName",
        "readOnly" : false
      },
      {
        "attribute" : "sn",
        "readOnly" : false
      },
      {
        "attribute" : "accountStatus",
        "readOnly" : true
      }
    ]
  } ]
}' \
"https://localhost:8443/openidm/internal/role/support"
{
  "_id": "support",
  "_rev": "00000000bfbac2ed",
  "name": "support",
  "description": "Support Role",
  "temporalConstraints": [],
  "condition": null,
  "privileges": [
    {
      "name": "support",
```

```
        "description": "Support access to user information.",
        "path": "managed/user",
        "permissions": [
          "VIEW",
          "UPDATE",
          "CREATE"
        ],
        "actions": [],
        "filter": null,
        "accessFlags": [
          {
            "attribute": "userName",
            "readOnly": false
          },
          {
            "attribute": "mail",
            "readOnly": false
          },
          {
            "attribute": "givenName",
            "readOnly": false
          },
          {
            "attribute": "sn",
            "readOnly": false
          },
          {
            "attribute": "accountStatus",
            "readOnly": true
          }
        ]
      }
    ]
  }
```

Policies related to privileges

When creating privileges, IDM runs policies found in `policy.json` and `policy.js`, including the five policies used for validating privileges:

`valid-accessFlags-object`

Verifies that `accessFlag` objects are correctly formatted. Only two fields are permitted in an `accessFlag` object: `readOnly`, which must be a boolean; and `attribute`, which must be a string.

`valid-array-items`

Verifies that each item in an array contains the properties specified in `policy.json`, and that each of those properties satisfies any specific policies applied to it. By default, this is used to verify that each privilege contains `name`, `path`, `accessFlags`, `actions`, and `permissions` properties, and that the `filter` property is valid if included.

`valid-permissions`

Verifies that the permissions set on the privilege are all valid and can be achieved with the `accessFlags` that have been set. It checks:

- `CREATE` permissions must have write access to all properties required to create a new object.
- `CREATE` and `UPDATE` permissions must have write access to at least one property.
- `ACTION` permissions must include a list of allowed actions, with at least one action included.
- If any attributes have write access, then the privilege must also have either `CREATE` or `UPDATE` permission.
- All permissions listed must be valid types of permission: `VIEW`, `CREATE`, `UPDATE`, `ACTION`, or `DELETE`. Also, no permissions are repeated.

`valid-privilege-path`

Verifies that the `path` specified in the privilege is a valid object with a schema for IDM to reference. Only objects with a schema (such as `managed/user`) can have privileges applied to them.

`valid-query-filter`

Verifies that the query filter used to filter privileges is a valid query.

For more information about policies and creating custom policies, refer to Use policies to validate data.

Get privileges on a resource

To determine which privileges a user has on a service, you can query the privilege endpoint for a given resource path or object, based on the user you are currently logged in as. For example, if a user is a member of the support role mentioned in the previous example, checking the user's privileges for the `managed/user` resource would look like this:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/privilege/managed/user"
{
  "VIEW": {
    "allowed": true,
    "properties": [
      "userName",
      "givenName",
      "sn",
      "mail",
      "accountStatus"
    ]
  },
  "CREATE": {
    "allowed": true,
    "properties": [
      "userName",
      "givenName",
      "sn",
      "mail"
    ]
  },
  "UPDATE": {
    "allowed": true,
    "properties": [
      "userName",
      "givenName",
      "sn",
      "mail"
    ]
  },
  "DELETE": {
    "allowed": false
  },
  "ACTION": {
    "allowed": false,
    "actions": []
  }
}
```

In the above example, `accountStatus` is listed as a property for `VIEW`, but not for `CREATE` or `UPDATE`, because the privilege sets this property to be read only. Since both `CREATE` and `UPDATE` need the ability to write to a property, setting `readOnly` to false applies to both permissions. If you need more granular control, split these permissions into two privileges.

In addition to checking privileges for a resource, it is also possible to check privileges for specific objects within a resource, such as `managed/user/scarter`.

Create a delegated administrator

You can use the IDM REST API to create an `internal/role` with privileges that have object, array, and relationship type attribute access. You can then use that role as a delegated administrator to perform operations on those attributes.

> **Note**
>
> If you want to experiment with delegated administrators in Postman, download and import this Postman collection.

Use the following example to create a delegated administrator:

To ensure a role object exists when roles are requested, you must create a managed role.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "name": "testManagedRole",
  "description": "a managed role for test"
}' \
"http://localhost:8080/openidm/managed/role/testManagedRole"
{
  "_id": "testManagedRole",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-181",
  "name": "testManagedRole",
  "description": "a managed role for test"
}
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd"
}' \
"http://localhost:8080/openidm/managed/user/?_action=create"
{
  "_id": "9cae97b7-3bf3-4107-96d5-39ad153629db",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1223",
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active",
  "effectiveRoles": [],
  "memberOfOrgIDs": [],
  "effectiveAssignments": []
}
```

In this step, you'll create two users with the following attributes:

- `preferences`
- `manager`
- `roles`

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "userName": "scarter",
  "sn": "Carter",
  "givenName": "Steven",
  "mail": "scarter@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "manager": {"_ref": "managed/user/9cae97b7-3bf3-4107-96d5-39ad153629db"},
  "roles": [{"_ref": "managed/role/testManagedRole"}]
}' \
"http://localhost:8080/openidm/managed/user/?_action=create"
{
  "_id": "917bc052-ef39-4add-ae05-0a278e2de9c0",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1238",
  "userName": "scarter",
  "sn": "Carter",
  "givenName": "Steven",
  "mail": "scarter@example.com",
  "telephoneNumber": "082082082",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "accountStatus": "active",
  "effectiveRoles": [
    {
      "_refResourceCollection": "managed/role",
      "_refResourceId": "testManagedRole",
      "_ref": "managed/role/testManagedRole"
    }
  ],
  "memberOfOrgIDs": [],
  "effectiveAssignments": []
}
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "userName": "jdoe",
  "sn": "Doe",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "manager": {"_ref": "managed/user/9cae97b7-3bf3-4107-96d5-39ad153629db"},
  "roles": [{"_ref": "managed/role/testManagedRole"}]
}' \
"http://localhost:8080/openidm/managed/user/?_action=create"
{
  "_id": "aca0042c-9f4c-4ad5-8cf7-aca0adeb3470",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1267",
  "userName": "jdoe",
  "sn": "Doe",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "082082082",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "accountStatus": "active",
  "effectiveRoles": [
    {
      "_refResourceCollection": "managed/role",
      "_refResourceId": "testManagedRole",
      "_ref": "managed/role/testManagedRole"
    }
  ],
  "memberOfOrgIDs": [],
  "effectiveAssignments": []
}
```

You will delegate an internal/role with privileges to this user in the next step:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "userName": "bjensen",
  "sn": "Jensen",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd"
}' \
"http://localhost:8080/openidm/managed/user/?_action=create"
{
  "_id": "2d726b2a-3324-44b3-ba40-91b154d4f51e",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1291",
  "userName": "bjensen",
  "sn": "Jensen",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active",
  "effectiveRoles": [],
  "memberOfOrgIDs": [],
  "effectiveAssignments": []
}
```

This role will have the following privileges:

- A `managed/user` privilege with accessFlags attributes that are of types: “String” , “boolean” , and “number” ; but also for:
  - An object type that is not a relationship (`preferences`).
  - An object type that is a relationship (`manager`).
  - Array types that are relationships (`roles` , `authzRoles` , `reports`).
- A `managed/role` privilege for viewing details of the “roles” property of a managed user.
- An `internal/role` privilege for viewing the details of the “authzRoles” property of a managed user.

Note

You can populate the privilege `filter` field to apply a finer level of permissions for what a delegated administrator can access or do with certain objects. The `filter` field is omitted in this example to allow all.
For properties that are *not* relationships, such as `preferences` , you can’t specify finer-grained permissions. For example, you can’t set permissions on `preferences/marketing` .

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "name": "internal_role_with_object_array_and_relationship_privileges",
  "description": "an internal role that has privileges for object & array types and relationships",
  "privileges": [
    {
      "name": "managed_user_privilege",
      "path": "managed/user",
      "permissions": [
        "VIEW",
        "CREATE",
        "UPDATE",
        "DELETE"
      ],
      "actions": [],
      "accessFlags": [
        {
          "attribute": "userName",
          "readOnly": false
        },
        {
          "attribute": "password",
          "readOnly": false
        },
        {
          "attribute": "givenName",
          "readOnly": false
        },
        {
          "attribute": "sn",
          "readOnly": false
        },
        {
          "attribute": "mail",
          "readOnly": false
        },
        {
          "attribute": "description",
          "readOnly": false
        },
        {
          "attribute": "accountStatus",
          "readOnly": false
        },
        {
          "attribute": "telephoneNumber",
          "readOnly": false
        },
        {
```

```
          "attribute": "postalAddress",
          "readOnly": false
        },
        {
          "attribute": "city",
          "readOnly": false
        },
        {
          "attribute": "postalCode",
          "readOnly": false
        },
        {
          "attribute": "country",
          "readOnly": false
        },
        {
          "attribute": "stateProvince",
          "readOnly": false
        },
        {
          "attribute": "preferences",
          "readOnly": false
        },
        {
          "attribute": "roles",
          "readOnly": false
        },
        {
          "attribute": "manager",
          "readOnly": false
        },
        {
          "attribute": "authzRoles",
          "readOnly": false
        },
        {
          "attribute": "reports",
          "readOnly": false
        }
      ]
    },
    {
      "name": "managed_role_privilege",
      "path": "managed/role",
      "permissions": [
        "VIEW"
      ],
      "actions": [],
      "accessFlags": [
        {
          "attribute": "name",
          "readOnly": true
        },
        {
          "attribute": "description",
```

```
            "readOnly": true
          }
        ]
      },
      {
        "name": "internal_role_privilege",
        "path": "internal/role",
        "permissions": [
          "VIEW"
        ],
        "actions": [],
        "accessFlags": [
          {
            "attribute": "name",
            "readOnly": true
          },
          {
            "attribute": "description",
            "readOnly": true
          },
          {
            "attribute": "authzMembers",
            "readOnly": true
          }
        ]
      }
    ]
}' \
"http://localhost:8080/openidm/internal/role/testInternalRole"
{
  "_id": "testInternalRole",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-300",
  "name": "internal_role_with_object_array_and_relationship_privileges",
  "description": "an internal role that has privileges for object & array types and relationships",
  "temporalConstraints": [],
  "condition": null,
  "privileges": [
    {
      "name": "managed_user_privilege",
      "path": "managed/user",
      "permissions": [
        "VIEW",
        "CREATE",
        "UPDATE",
        "DELETE"
      ],
      "actions": [],
      "accessFlags": [
        {
          "attribute": "userName",
          "readOnly": false
        },
        {
          "attribute": "password",
          "readOnly": false
```

```
        },
        {
          "attribute": "givenName",
          "readOnly": false
        },
        {
          "attribute": "sn",
          "readOnly": false
        },
        {
          "attribute": "mail",
          "readOnly": false
        },
        {
          "attribute": "description",
          "readOnly": false
        },
        {
          "attribute": "accountStatus",
          "readOnly": false
        },
        {
          "attribute": "telephoneNumber",
          "readOnly": false
        },
        {
          "attribute": "postalAddress",
          "readOnly": false
        },
        {
          "attribute": "city",
          "readOnly": false
        },
        {
          "attribute": "postalCode",
          "readOnly": false
        },
        {
          "attribute": "country",
          "readOnly": false
        },
        {
          "attribute": "stateProvince",
          "readOnly": false
        },
        {
          "attribute": "preferences",
          "readOnly": false
        },
        {
          "attribute": "roles",
          "readOnly": false
        },
        {
          "attribute": "manager",
```

```
            "readOnly": false
          },
          {
            "attribute": "authzRoles",
            "readOnly": false
          },
          {
            "attribute": "reports",
            "readOnly": false
          }
        ]
      },
      {
        "name": "managed_role_privilege",
        "path": "managed/role",
        "permissions": [
          "VIEW"
        ],
        "actions": [],
        "accessFlags": [
          {
            "attribute": "name",
            "readOnly": true
          },
          {
            "attribute": "description",
            "readOnly": true
          }
        ]
      },
      {
        "name": "internal_role_privilege",
        "path": "internal/role",
        "permissions": [
          "VIEW"
        ],
        "actions": [],
        "accessFlags": [
          {
            "attribute": "name",
            "readOnly": true
          },
          {
            "attribute": "description",
            "readOnly": true
          },
          {
            "attribute": "authzMembers",
            "readOnly": true
          }
        ]
      }
    ]
}
```

In this step, assign the internal/role from step 5 to the user created in step 4 by creating a relationship:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "_ref": "managed/user/2d726b2a-3324-44b3-ba40-91b154d4f51e",
  "_refProperties": {}
}' \
"http://localhost:8080/openidm/internal/role/testInternalRole/authzMembers?_action=create"
{
  "_id": "2e21f423-f934-4ed7-b6fd-9883b69d52d8",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1304",
  "_ref": "managed/user/2d726b2a-3324-44b3-ba40-91b154d4f51e",
  "_refResourceCollection": "managed/user",
  "_refResourceId": "2d726b2a-3324-44b3-ba40-91b154d4f51e",
  "_refProperties": {
    "_id": "2e21f423-f934-4ed7-b6fd-9883b69d52d8",
    "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1304"
  }
}
```

You can now perform operations as a delegated administrator, such as:

The query results display all users' properties that are allowed by the privileges:

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true&_pageSize=100&_fields=*,*_ref/*"
{
  "result": [
    {
      "_id": "9cae97b7-3bf3-4107-96d5-39ad153629db",
      "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1223",
      "userName": "psmith",
      "sn": "Smith",
      "givenName": "Patricia",
      "mail": "psmith@example.com",
      "telephoneNumber": "082082082",
      "accountStatus": "active",
      "reports": [
        {
          "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1267",
          "_id": "aca0042c-9f4c-4ad5-8cf7-aca0adeb3470",
          "userName": "jdoe",
          "sn": "Doe",
          "givenName": "John",
          "mail": "jdoe@example.com",
          "telephoneNumber": "082082082",
          "preferences": {
            "updates": true,
            "marketing": false
          },
          "accountStatus": "active",
          "_ref": "managed/user/aca0042c-9f4c-4ad5-8cf7-aca0adeb3470",
          "_refResourceCollection": "managed/user",
          "_refResourceId": "aca0042c-9f4c-4ad5-8cf7-aca0adeb3470",
          "_refProperties": {
            "_id": "e01a922b-a60d-46c2-b6bc-2b821c1580b4",
            "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1262"
          }
        },
        {
          "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1238",
          "_id": "917bc052-ef39-4add-ae05-0a278e2de9c0",
          "userName": "scarter",
          "sn": "Carter",
          "givenName": "Steven",
          "mail": "scarter@example.com",
          "telephoneNumber": "082082082",
          "preferences": {
            "updates": true,
            "marketing": false
          },
          "accountStatus": "active",
          "_ref": "managed/user/917bc052-ef39-4add-ae05-0a278e2de9c0",
          "_refResourceCollection": "managed/user",
          "_refResourceId": "917bc052-ef39-4add-ae05-0a278e2de9c0",
```

```
          "_refProperties": {
            "_id": "5bc2c633-8ae1-4ea2-adf6-8aa7ce5f8e70",
            "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1233"
          }
        }
      ],
      "manager": null,
      "roles": [],
      "authzRoles": []
    },
    {
      "_id": "917bc052-ef39-4add-ae05-0a278e2de9c0",
      "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1238",
      "userName": "scarter",
      "sn": "Carter",
      "givenName": "Steven",
      "mail": "scarter@example.com",
      "telephoneNumber": "082082082",
      "preferences": {
        "updates": true,
        "marketing": false
      },
      "accountStatus": "active",
      "reports": [],
      "manager": {
        "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1223",
        "_id": "9cae97b7-3bf3-4107-96d5-39ad153629db",
        "userName": "psmith",
        "sn": "Smith",
        "givenName": "Patricia",
        "mail": "psmith@example.com",
        "telephoneNumber": "082082082",
        "accountStatus": "active",
        "_ref": "managed/user/9cae97b7-3bf3-4107-96d5-39ad153629db",
        "_refResourceCollection": "managed/user",
        "_refResourceId": "9cae97b7-3bf3-4107-96d5-39ad153629db",
        "_refProperties": {
          "_id": "5bc2c633-8ae1-4ea2-adf6-8aa7ce5f8e70",
          "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1233"
        }
      },
      "roles": [
        {
          "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-181",
          "_id": "testManagedRole",
          "name": "testManagedRole",
          "description": "a managed role for test",
          "_ref": "managed/role/testManagedRole",
          "_refResourceCollection": "managed/role",
          "_refResourceId": "testManagedRole",
          "_refProperties": {
            "_id": "a33e2de0-83ff-481c-b8a7-8ffbc02d135c",
            "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1235"
          }
        }
```

```
      ],
      "authzRoles": []
    },
    {
      "_id": "aca0042c-9f4c-4ad5-8cf7-aca0adeb3470",
      "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1267",
      "userName": "jdoe",
      "sn": "Doe",
      "givenName": "John",
      "mail": "jdoe@example.com",
      "telephoneNumber": "082082082",
      "preferences": {
        "updates": true,
        "marketing": false
      },
      "accountStatus": "active",
      "reports": [],
      "manager": {
        "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1223",
        "_id": "9cae97b7-3bf3-4107-96d5-39ad153629db",
        "userName": "psmith",
        "sn": "Smith",
        "givenName": "Patricia",
        "mail": "psmith@example.com",
        "telephoneNumber": "082082082",
        "accountStatus": "active",
        "_ref": "managed/user/9cae97b7-3bf3-4107-96d5-39ad153629db",
        "_refResourceCollection": "managed/user",
        "_refResourceId": "9cae97b7-3bf3-4107-96d5-39ad153629db",
        "_refProperties": {
          "_id": "e01a922b-a60d-46c2-b6bc-2b821c1580b4",
          "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1262"
        }
      },
      "roles": [
        {
          "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-181",
          "_id": "testManagedRole",
          "name": "testManagedRole",
          "description": "a managed role for test",
          "_ref": "managed/role/testManagedRole",
          "_refResourceCollection": "managed/role",
          "_refResourceId": "testManagedRole",
          "_refProperties": {
            "_id": "1528ab24-3ec3-4113-ac3f-26cc71a2d678",
            "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1264"
          }
        }
      ],
      "authzRoles": []
    },
    {
      "_id": "2d726b2a-3324-44b3-ba40-91b154d4f51e",
      "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1305",
      "userName": "bjensen",
```

```
      "sn": "Jensen",
      "givenName": "Barbara",
      "mail": "bjensen@example.com",
      "telephoneNumber": "082082082",
      "accountStatus": "active",
      "reports": [],
      "manager": null,
      "roles": [],
      "authzRoles": [
        {
          "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-300",
          "_id": "testInternalRole",
          "name": "internal_role_with_object_array_and_relationship_privileges",
          "description": "an internal role that has privileges for object & array types and relationships",
          "_ref": "internal/role/testInternalRole",
          "_refResourceCollection": "internal/role",
          "_refResourceId": "testInternalRole",
          "_refProperties": {
            "_id": "2e21f423-f934-4ed7-b6fd-9883b69d52d8",
            "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1304"
          }
        }
      ]
    }
  ],
  "resultCount": 4,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```


```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--request GET \
"http://localhost:8080/openidm/managed/user/aca0042c-9f4c-4ad5-8cf7-aca0adeb3470?_fields=preferences"
```


```
{
  "_id": "aca0042c-9f4c-4ad5-8cf7-aca0adeb3470",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1267",
  "preferences": {
    "updates": true,
    "marketing": false
  }
}
```

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--request GET \
"http://localhost:8080/openidm/managed/user/917bc052-ef39-4add-ae05-0a278e2de9c0/roles?
_queryFilter=true&_fields=*"
{
  "result": [
    {
      "_id": "a33e2de0-83ff-481c-b8a7-8ffbc02d135c",
      "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1235",
      "name": "testManagedRole",
      "description": "a managed role for test",
      "_refResourceCollection": "managed/role",
      "_refResourceId": "testManagedRole",
      "_refResourceRev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-181",
      "_ref": "managed/role/testManagedRole",
      "_refProperties": {
        "_id": "a33e2de0-83ff-481c-b8a7-8ffbc02d135c",
        "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1235"
      }
    }
  ],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--request GET \
"http://localhost:8080/openidm/managed/user/917bc052-ef39-4add-ae05-0a278e2de9c0/manager?_fields=*"
{
  "_id": "5bc2c633-8ae1-4ea2-adf6-8aa7ce5f8e70",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1233",
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active",
  "_refResourceCollection": "managed/user",
  "_refResourceId": "9cae97b7-3bf3-4107-96d5-39ad153629db",
  "_refResourceRev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1223",
  "_ref": "managed/user/9cae97b7-3bf3-4107-96d5-39ad153629db",
  "_refProperties": {
    "_id": "5bc2c633-8ae1-4ea2-adf6-8aa7ce5f8e70",
    "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1233"
  }
}
```

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
   "operation" : "replace",
   "field" : "reports",
   "value" : [{"_ref" : "managed/user/917bc052-ef39-4add-ae05-0a278e2de9c0"}]
} ]' \
"http://localhost:8080/openidm/managed/user/9cae97b7-3bf3-4107-96d5-39ad153629db"
{
  "_id": "9cae97b7-3bf3-4107-96d5-39ad153629db",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1223",
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active"
}
```

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "manager",
    "value": {"_ref" : "managed/user/9cae97b7-3bf3-4107-96d5-39ad153629db"}
  }
]' \
"http://localhost:8080/openidm/managed/user/aca0042c-9f4c-4ad5-8cf7-aca0adeb3470"
{
  "_id": "aca0042c-9f4c-4ad5-8cf7-aca0adeb3470",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1517",
  "userName": "jdoe",
  "sn": "Doe",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "082082082",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "accountStatus": "active"
}
```

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
     "operation": "remove",
     "field": "manager"
  }
]' \
"http://localhost:8080/openidm/managed/user/aca0042c-9f4c-4ad5-8cf7-aca0adeb3470"
{
  "_id": "aca0042c-9f4c-4ad5-8cf7-aca0adeb3470",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1545",
  "userName": "jdoe",
  "sn": "Doe",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "082082082",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "accountStatus": "active"
}
```

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "replace",
    "field": "manager",
    "value": {"_ref" : "managed/user/aca0042c-9f4c-4ad5-8cf7-aca0adeb3470"}
  }
]' \
"http://localhost:8080/openidm/managed/user/917bc052-ef39-4add-ae05-0a278e2de9c0"
{
  "_id": "917bc052-ef39-4add-ae05-0a278e2de9c0",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1565",
  "userName": "scarter",
  "sn": "Carter",
  "givenName": "Steven",
  "mail": "scarter@example.com",
  "telephoneNumber": "082082082",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "accountStatus": "active"
}
```

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Content-Type: application/json" \
--request DELETE \
"http://localhost:8080/openidm/managed/user/9cae97b7-3bf3-4107-96d5-39ad153629db"
{
  "_id": "9cae97b7-3bf3-4107-96d5-39ad153629db",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-1223",
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active"
}
```

- Using POST:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--request POST \
--data '{
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd"
}' \
"http://localhost:8080/openidm/managed/user"
{
  "_id": "1a20930b-cf61-4b43-a730-9f73af9147cb",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-571",
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active"
}
```

- Using PUT:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd"
}' \
"http://localhost:8080/openidm/managed/user/psmith"
{
  "_id": "psmith",
  "_rev": "200bc5d6-7cc1-4648-a854-3137f3d9c103-590",
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active"
}
```

Warning

Delegated administration may not work as expected when using DS as your repository if `_id` is something *other* than a UUID. JDBC repositories may use other forms for `_id`, though using a UUID is still recommended.

Note

For more examples, including working with filters, refer to the Postman collection.

Note

All patches are done with a PATCH request. Delegated administrator operations do not currently support using POST actions for patch requests (POST `_action=patch` will not work).

Warning

This process only applies to managed users. It will not work for internal users, as they cannot have roles.

Configure search UI for delegated administrators

The IDM End User UI support for delegated administration includes a search feature to filter a list of results. To help keep search performant when working with large lists, you can configure the following constraints:

- Set a minimum filter length.
- Disable the ability to filter and sort.

Note

These settings only affect delegated administrators.

Minimum filter search length

You can set `minimumUIFilterLength` in `conf/ui-configuration.json` to define when results start filtering:

```
"platformSettings" : {
    "managedObjectsSettings" : {
        "user" : {
            "minimumUIFilterLength" : 3
        }
    }
}
```

This setting prevents the UI from filtering until the user has typed at least three characters. `minimumUIFilterLength` can be used with any managed object, for example:

```
"platformSettings" : {
    "managedObjectsSettings" : {
        "user" : {
            "minimumUIFilterLength" : 3
        },
        "role" : {
            "minimumUIFilterLength" : 2
        }
    }
}
```

Disable sort and filter for resource collections

A *resource collection* is a set of objects that have a relationship with one or more other objects. For example:

- All users with a particular role assignment.
- All users who are members of an organization.

You can disable the ability to sort and filter resource collections using `"disableRelationshipSortAndSearch" : true` in `conf/ui-configuration.json`. This can be beneficial when working with *very* large lists. For example:

```
"platformSettings" : {
    "managedObjectsSettings" : {
        "user" : {
            "disableRelationshipSortAndSearch" : true,
            "minimumUIFilterLength" : 3
        }
    }
}
```

Authentication and session module configuration

This appendix includes configuration details for the authentication modules described in Authentication and Session Modules.

Authentication modules, as configured in the `authentication.json` file, include a number of properties.

Session Module

Authentication Property	Property as Listed in the Admin UI	Description
`keyAlias`	(not shown)	Used by the Jetty Web server to service SSL requests.
`maxTokenLifeMinutes`	Max Token Life (in seconds)	Maximum time before a session is cancelled. Note the different units for the property and the UI.

Authentication Property	Property as Listed in the Admin UI	Description
`tokenIdleTimeMinutes`	Token Idle Time (in seconds)	Maximum time before an idle session is cancelled. Note the different units for the property and the UI.
`sessionOnly`	Session Only	Whether the session continues after browser restarts.

Static User Module

Authentication Property	Property as Listed in the Admin UI	Description
`enabled`	Module Enabled	Does IDM use the module?
`queryOnResource`	Query on Resource	Endpoint hard coded to user `anonymous`
`username`	Static User Name	Default for the static user, `anonymous`
`password`	Static User Password	Default for the static user, `anonymous`
`defaultUserRoles`	Static User Role	Normally set to `openidm-reg` for self-registration

The following table applies to several authentication modules:

- Managed User
- Internal User
- Client Cert
- Passthrough
- IWA

The IWA module includes several Kerberos-related properties listed at the end of the table.

Common Module Properties

Authentication Property	Property as Listed in the Admin UI	Description
`enabled`	Module Enabled	Does IDM use the module?
`queryOnResource`	Query on Resource	Endpoint to query

Authentication Property	Property as Listed in the Admin UI	Description
`queryId`	Use Query ID	A defined `queryId` searches against the `queryOnResource` endpoint. An undefined `queryId` searches against `queryOnResource` with `action=reauthenticate`
`defaultUserRoles`	Default User Roles	Normally blank for managed users
`authenticationId`	Authentication ID	Defines how account credentials are derived from a `queryOnResource` endpoint
`userCredential`	User Credential	Defines how account credentials are derived from a `queryOnResource` endpoint; if required, typically `password` or `userPassword`
`userRoles`	User Roles	Defines how account roles are derived from a `queryOnResource` endpoint
`groupMembership`	Group Membership	Provides more information for calculated roles
`groupRoleMapping`	Group Role Mapping	Provides more information for calculated roles
`groupComparisonMethod`	Group Comparison Method	Provides more information for calculated roles
`augmentSecurityContext`	Augment Security Context	Includes a script that is executed only after a successful authentication request. For more information on this property, refer to Authenticate as a different user.
`servicePrincipal`	Kerberos Service Principal	(IWA only) For more information, refer to IWA
`keytabFileName`	Keytab File Name	(IWA only) For more information, refer to IWA
`kerberosRealm`	Kerberos Realm	(IWA only) For more information, refer to IWA
`kerberosServerName`	Kerberos Server Name	(IWA only) For more information, refer to IWA

Synchronization

Configure synchronization between ForgeRock® Identity Management and other resources.

Synchronizing identity data between resources is one of the core services of ForgeRock Identity Management (IDM). In this guide, you will learn about the different types of synchronization, and how to configure the flexible synchronization mechanism. This guide is written for systems integrators building solutions based on ForgeRock Identity Management services.

Synchronization overview

Understand synchronization types and configuration.

Mappings

Map data between resources.

Situations and actions

Learn about synchronization situations and how to configure actions for each.

Filter synchronization data

Use filtering mechanisms to limit the synchronized data.

Implicit sync and liveSync

Configure automatic synchronization between resources.

Reconciliation performance

Learn about ways to improve reconciliation performance.

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

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

Synchronization overview

Synchronization keeps data consistent across disparate resources. Within IDM, we refer to two resource types—*managed resources* (stored in the IDM repository) and *external resources*. An external resource can be any system that holds identity data, such as ForgeRock Directory Services (DS), Active Directory, a CSV file, a JDBC database, and so on.

IDM connects to external resources through *connectors*. Synchronization across resources happens when managed resources change or when IDM discovers a change on a system resource. There are various synchronization mechanisms that ensure data consistency.

Synchronization types

- IDM discovers and synchronizes changes from external resources by using *reconciliation* and *liveSync*.
- IDM synchronizes changes made to managed resources by using *reconciliation* and *implicit synchronization*.

Reconciliation

Reconciliation is the process of ensuring that the objects in two different data stores are consistent. Traditionally, reconciliation applies mainly to user objects, but IDM can reconcile any objects, such as groups, roles, and devices.

In any reconciliation operation, there is a *source system* (the system that contains the changes) and a *target system* (the system to which the changes will be propagated). The source and target system are defined in a *mapping*. The IDM repository can be either the source or the target in a mapping. You can configure multiple mappings for one IDM instance, depending on the external resources to which you are connecting.

To perform reconciliation, IDM analyzes both the source system *and* the target system, to discover the differences between them. 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 very thorough. It recognizes system error conditions and catches changes that might be missed by liveSync, and therefore serves as the basis for compliance and reporting.

LiveSync

LiveSync captures the changes that occur on an external system, and pushes those changes to IDM. IDM uses any defined mappings to replay those changes where they are required—to its managed objects, to another remote system, or to both. Unlike reconciliation, liveSync uses a polling system, and is intended to react quickly to changes as they happen.

To perform this polling, liveSync relies on a change detection mechanism on the external resource to determine which objects have changed. The change detection mechanism is specific to the external resource, and can be a time stamp, a sequence number, a change vector, or any other method of recording changes that have occurred on the system. For example, ForgeRock Directory Services (DS) implements a change log that provides IDM with a list of objects that have changed since the last request. Active Directory implements a change sequence number, and certain databases might have a `lastChange` attribute.

Implicit synchronization

Implicit synchronization automatically pushes changes that are made to IDM managed objects out to external systems.

For direct changes to managed objects, IDM immediately synchronizes those changes to all mappings configured to use those objects as their source. 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.

> **Note**
>
> Implicit synchronization only synchronizes *changed objects* to external resources. To synchronize a complete data set, you must run a reconciliation operation. The entire changed object is synchronized. If you want to synchronize only the attributes that have changed, you can modify the `onUpdate` script in your mapping to compare attribute values before pushing changes.

Synchronization configuration overview

This section describes the high-level steps required to set up synchronization between two resources. A basic synchronization configuration involves the following steps:

1. Set up a connection between the source and target resource.

   A connector configuration references a specific connector type and indicates the connection details of the external resource. You must define a connector configuration for each external resource to which you are connecting.

   For more information, refer to Connections between resources.

2. Map source objects to target objects.

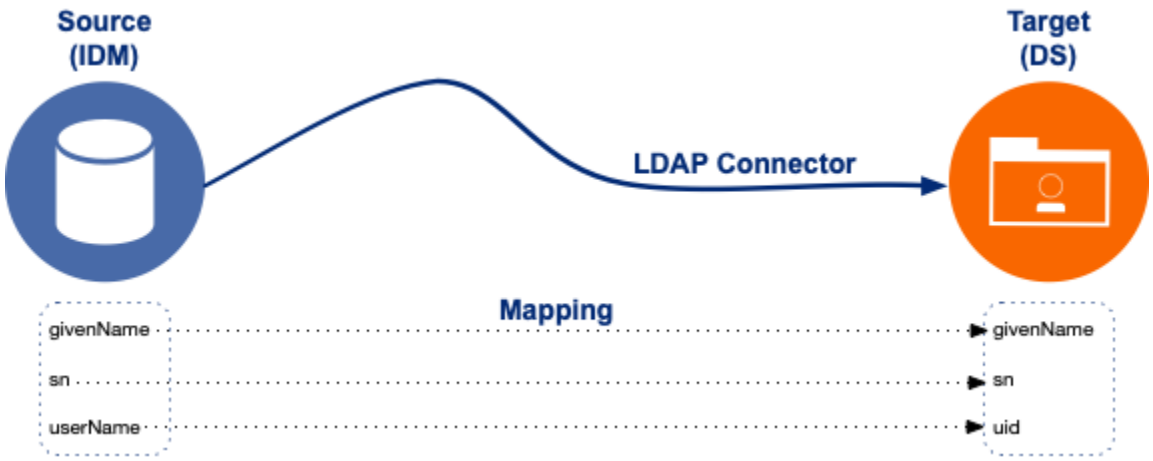
   The mapping configuration your project's `conf/sync.json` file or in individual mapping files. Mappings are synchronized in the order in which they are specified in the `sync.json` file. If there are multiple mapping files, the `syncAfter` property dictates the order in which they are processed.

   For more information, refer to Resource mapping.

3. Configure any scripts that are required to check source and target objects, and to manipulate attributes.

4. In addition to these configuration elements, IDM stores a `links` table in its repository. The links table maintains a record of relationships established between source and target objects.

The following diagram illustrates the high-level synchronization configuration:



Data mapping model

IDM uses mappings to determine which data to synchronize, and how that data must be synchronized.

In general, identity management software implements one of the following data models:

- A meta-directory data model, where all data are mirrored in a central repository.

  The meta-directory model offers fast access at the risk of getting outdated data.

- A virtual data model, where only a minimum set of attributes are stored centrally, and most are loaded on demand from the external resources in which they are stored.

  The virtual model guarantees fresh data, but pays for that guarantee in terms of performance.

IDM leaves the data model choice up to you. You determine the right trade-offs for a particular deployment. IDM 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 IDM dynamically updates the repository to store the managed object attributes that you configure.

Connections between resources

A connector lets you transfer data between different resource systems. The connector configuration works in conjunction with the synchronization mapping and specifies how target object attributes map to attributes on external objects.

Connector configuration files exist in your project's conf directory and are named `provisioner.resource-name.json`, where resource-name reflects the connector technology and the external resource. For example, `openicf-csv`.

To create and modify connector configurations, use one of the following methods:

Configure connectors using the admin UI

From the navigation bar, click **Configure > Connectors**, and do one of the following.

- Select an existing connector to modify.
- Click **New Connector**, and configure the new connector.

Edit connector configuration files

IDM provides a number of sample provisioner files in the `path/to/openidm/samples/example-configurations/provisioners` directory. To modify connector configuration files directly, edit one of the sample provisioner files that corresponds to the resource to which you are connecting.

The following excerpt of an example LDAP connector configuration shows the attributes of an account object type. In the attribute mapping definitions, the attribute name is mapped from the IDM managed object to the `nativeName` (the attribute name used on the external resource). The `lastName` attribute in IDM is mapped to the `sn` attribute in LDAP. The `homePhone` attribute is defined as an array, because it can have multiple values:

```
{
    ...
    "objectTypes": {
        "account": {
            "lastName": {
                "type": "string",
                "required": true,
                "nativeName": "sn",
                "nativeType": "string"
            },
            "homePhone": {
                "type": "array",
                "items": {
                    "type": "string",
                    "nativeType": "string"
                },
                "nativeName": "homePhone",
                "nativeType": "string"
            }
        }
    }
}
```

For IDM to access external resource objects and attributes, the object and its attributes must match the connector configuration. Note that the connector file only maps IDM managed objects and attributes to their counterparts on the external resource. To construct attributes and to manipulate their values, you use a synchronization mapping, described in Resource mapping.

Configure connectors using REST

Create connector configurations using REST with the `createCoreConfig` and `createFullConfig` actions. For more information, refer to Configure Connectors Using REST.

Resource mapping

A synchronization mapping specifies a relationship between objects and their attributes in two data stores. The following example shows a typical attribute mapping, between objects in an external LDAP directory and an IDM managed user data store:

```
"source": "lastName",
"target": "sn"
```

In this case, the `lastName` source attribute is mapped to the `sn` (surname) attribute in the target LDAP directory.

The core synchronization configuration is defined in the mapping configuration.

You can define a single file with all your mappings (`conf/sync.json`), or a separate file per mapping. Individual mapping files are named `mapping-mappingName.json`; for example, `mapping-managedUser_systemCsvfileAccounts.json`. Individual mapping files can be useful if your deployment includes many mappings that are difficult to manage in a single file. You can also use a combination of individual mapping files and a monolithic `sync.json` file, particularly if you are adding mappings to an existing deployment.

If you use a single `sync.json` file, mappings are processed in the order in which they appear within that file. If you use multiple mapping files, mappings are processed according to the `syncAfter` property in the mapping. The following example indicates that this particular mapping must be processed after the `managedUser_systemCsvfileAccount` mapping:

```
"source" : "managed/user",
"target" : "system/csvfile/account",
"syncAfter" : [ "managedUser_systemCsvfileAccount" ],
```

If you use a combination of `sync.json` and individual mapping files, the synchronization engine processes the mappings in `sync.json` first (in order), and then any mappings specified in the individual mapping files, according to the `syncAfter` property in each mapping.

For a list of *all* mappings, use the following request:

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

This call returns the mappings in the order in which they will be processed.

> **Note**
>
> The admin UI only shows the mappings configured in the `sync.json` file. Don't use the admin UI to add or change mappings in individual mapping files.

Mappings are always defined from a *source* resource to a *target* resource. To configure bidirectional synchronization, you must define two mappings. For example, to configure bidirectional synchronization between an LDAP server and an IDM repository, you would define the following two mappings:

- LDAP Server > IDM Repository
- IDM Repository > LDAP Server

Bidirectional mappings can include a `links` property that lets you reuse the links established between objects, for both mappings. For more information, refer to Reuse Links Between Mappings.

You can update a mapping while the server is running. To avoid inconsistencies between data stores, don't update a mapping while a reconciliation is in progress *for that mapping*.

Specifying default fields

The `defaultSourceFields` and `defaultTargetFields` optional fields allow more control over which attributes are fetched during read operations during synchronization and reconciliation.

When to specify

1. During synchronization reads: If IDM needs to read a source or target object while executing a synchronization operation because the object isn't already loaded in memory for that specific task, it will include the attributes listed in `defaultSourceFields` or `defaultTargetFields` in the read request.

   > **Important**
   >
   > This is an efficiency feature. If an object was already loaded into memory earlier in the same overall transaction (for instance, during reconciliation) before the sync operation needs it, IDM won't automatically reread the object to add the attributes specified in these default fields. The fields are primarily used when a *new* read is initiated during the sync process itself.

2. During reconciliation queries: If you *don't* specify an explicit list of attributes in the `sourceQuery` or `targetQuery` configuration, IDM will fall back to using `defaultSourceFields` or `defaultTargetFields` respectively. This can be more helpful than the older behavior, where some default queries might only retrieve the object's `_id`.

If you define a list of attributes within the `sourceQuery` or `targetQuery` settings, that specific list always takes precedence. The `defaultSourceFields` and `defaultTargetFields` only act as defaults when no specific list is provided in those contexts.

When not to specify

There are a couple of situations where these fields won't dictate the attributes loaded for a source object during a synchronization operation:

- Implicit synchronization: In implicit sync scenarios, the source object that triggered the event is usually already loaded, so IDM doesn't perform a new read using `defaultSourceFields`.
- Object already loaded using explicit query: If the source or target object was already fully loaded during reconciliation because specific attributes were requested using the higher-priority `sourceQuery` or `targetQuery` settings, then `defaultSourceFields` or `defaultTargetFields` won't cause another read.

Configure a resource mapping

Objects in external resources are specified in a mapping as `system/name/object-type`, where name is the name used in the connector configuration, and object-type is the object defined in the connector configuration list of object types. Objects in the repository are specified in the mapping as `managed/object-type`, where object-type is defined in the managed object configuration.

External resources, and IDM managed objects, can be the *source* or the *target* in a mapping. By convention, the mapping name is a string of the form `source_target`, as shown in the following example:

```
{
    "mappings": [
        {
            "name": "systemLdapAccounts_managedUser",
            "source": "system/ldap/account",
            "target": "managed/user",
            "properties": [
                {
                    "source": "lastName",
                    "target": "sn"
                },
                {
                    "source": "telephoneNumber",
                    "target": "telephoneNumber"
                },
                {
                    "target": "phoneExtension",
                    "default": "0047"
                },
                {
                    "source": "email",
                    "target": "mail",
                    "comment": "Set mail if non-empty.",
                    "condition": {
                        "type": "text/javascript",
                        "source": "(object.email != null)"
                    }
                },
                {
                    "source": "",
                    "target": "displayName",
                    "transform": {
                        "type": "text/javascript",
                        "source": "source.lastName +', ' + source.firstName;"
                    }
                },
               {
                    "source" : "uid",
                    "target" : "userName",
                    "condition" : "/linkQualifier eq \"user\""
                    }
               },
            ]
        }
    ]
}
```

In this example, the name of the source is the external resource (`ldap`), and the target is IDM's user repository; specifically, `managed/user`. The `properties` defined in the mapping correspond to attribute names that are defined in the IDM configuration. For example, the source attribute `uid` is defined in the `ldap` connector configuration file, rather than on the external resource itself.

Individual mapping files do not include a `name` property. The mapping `name` is taken from the file name. For example, the mapping shown in Basic LDAP Mapping would be in a file named `mapping-systemLdapAccounts_managedUser.json`, and start as follows:

```
{
   "source": "system/ldap/account",
   "target": "managed/user",
   ...
}
```

Configure mappings using the admin UI

To set up a synchronization mapping using the admin UI:

1. From the navigation bar, click **Configure > Mappings**.
2. Click **New Mapping**.
3. On the **New Mapping** page, select a source and target resource from the configured resources at the bottom of the window, and click **Create Mapping**.

   You can filter these resources to display only connector configurations or managed objects.
4. Select **Add property** on the **Attributes** grid to map a target property to its corresponding source property.

   The **Property** list shows all configured properties on the target resource. If the target resource is specified in a connector configuration, the **Property** list shows all properties configured for this connector. If the target resource is a managed object, the **Property** list shows the list of properties (defined in the managed object configuration for that object).

   > **Tip**
   >
   > - Select **Add Missing Required Properties** to add all the properties that are configured as ***required*** on the target resource. You can then map these required properties individually.
   > - Select **Quick Mapping** to show all source and target properties simultaneously. Drag a source property onto its corresponding target property, or vice versa. When you're done, click **Save**.
5. To test your mapping configuration on a single source entry, click the **Behaviors** tab and scroll down to **Single Record Reconciliation**. Search for the entry to reconcile.

   The UI displays a preview of the target entry after a reconciliation. You can then click **Reconcile Selected Record** to perform the reconciliation on that one source entry.

Remove a mapping

- To remove a mapping, delete the corresponding section from your mapping configuration. If you have configured mappings in individual mapping files, delete the file associated with the mapping you want to remove.
- To remove a mapping using the admin UI, select **Configure > Mappings**, and then click **Delete** under the mapping to remove.

> **Note**
>
> If you delete a mapping using the admin UI, the `delete-mapping-links` script removes all links related to that mapping from the repository. If you delete the mapping directly in the configuration file, no links are deleted from the repository.

Transform attributes using a mapping

You can use a mapping to define attribute transformations during synchronization. In the following sample mapping excerpt, the value of the `displayName` attribute on the target is set using a combination of the `lastName` and `firstName` attribute values from the source:

```
{
    "source": "",
    "target": "displayName",
    "transform": {
        "type": "text/javascript",
        "source": "source.lastName +', ' + source.firstName;"
    }
},
```

For transformations, the `source` property is optional. However, a source object is only available if you specify the `source` property. Therefore, in order to use `source.lastName` and `source.firstName` to calculate the `displayName`, the example specifies `"source" : ""`.

If you set `"source" : ""` (not specifying an attribute), the entire object is regarded as the source, and you must include the attribute name in the transformation script. For example, to transform the source username to lowercase, your script would be `source.mail.toLowerCase();`. If you do specify a source attribute (for example, `"source" : "mail"`), just that attribute is regarded as the source. In this case, the transformation script would be `source.toLowerCase();`.

Configure attribute transformation using the admin UI

1. From the navigation bar, click **Configure > Mappings**, and select a mapping.
2. Select the line with the target attribute value to set.
3. On the **Transformation Script** tab, select **Javascript** or **Groovy**, and enter the transformation as an **Inline Script**, or specify the path to the file containing your transformation script.

When you use the UI to map a property with an encrypted value, you are prompted to set up a transformation script to decrypt the value when that property is synchronized. The resulting mapping looks similar to the following, which shows the transformation of a user's `password` property:

```
{
    "target" : "userPassword",
    "source" : "password",
    "transform" : {
        "type" : "text/javascript",
        "globals" : { },
        "source" : "openidm.decrypt(source);"
    },
    "condition" : {
        "type" : "text/javascript",
        "globals" : { },
        "source" : "object.password != null"
    }
}
```

Default attribute values in a mapping

You can use a mapping to *create* attributes on the target resource. The following mapping excerpt creates a `phoneExtension` attribute with a default value of `0047` on the target object:

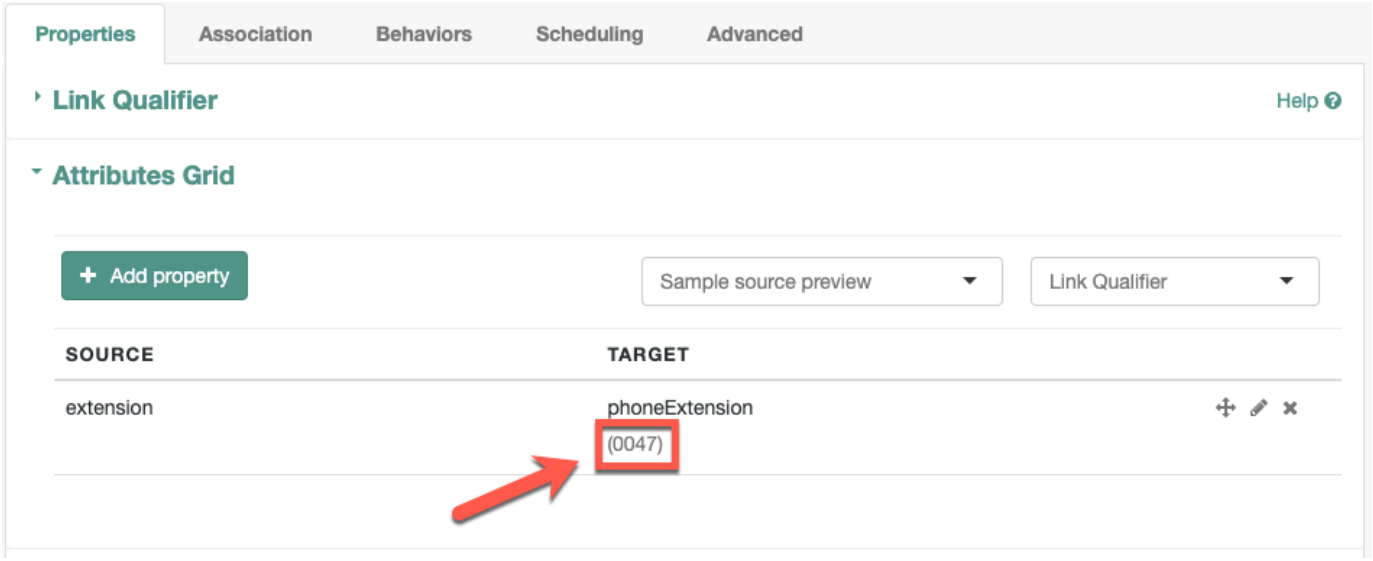
```
{
    "target": "phoneExtension",
    "default": "0047"
},
```

The `default` property specifies a value to assign to the attribute on the target object. Before IDM determines the value of the target attribute, it evaluates any applicable conditions, followed by any transformation scripts. If the `source` property and the `transform` script yield a null value, IDM applies the default value in the create and update actions. The default value overrides the target value, if one exists.

Configure default attribute values using the admin UI

1. From the navigation bar, click **Configure > Mappings**, and click the mapping to edit.
2. Click the **Properties** tab.
3. Expand the **Attributes Grid** node, and click the **Target** property to edit.
4. In the **Target Property: name** window, click the **Default Values** tab, and add or edit the default values.
5. Click **Save**.

   The default value displays in the **Attributes Grid**.



Scriptable conditions in a mapping

By default, IDM synchronizes all attributes in a mapping. For more complex relationships between source and target objects, you can define conditions under which IDM maps certain attributes. You can define two types of mapping conditions:

- *Scriptable conditions*, in which an attribute is mapped only if the defined script evaluates to `true`.
- *Condition filters*, a declarative filter that sets the conditions under which the attribute is mapped. Condition filters can include a *link qualifier* , that identifies the *type* of relationship between the source object and multiple target objects. For more information, refer to Map a Single Source Object to Multiple Target Objects.

  The following list shows examples of condition filters:

  - `"condition": "/object/country eq 'France'"` —Only map the attribute if the object's `country` attribute equals `France`.
  - `"condition": "/object/password pr"` —Only map the attribute if the object's `password` attribute is present.
  - `"condition": "/linkQualifier eq 'admin'"` —Only map the attribute if the link between this source and target object is of type `admin`.

Configure mapping conditions using the admin UI

1. From the navigation bar, click **Configure > Mappings**, and click the mapping to edit.
2. Click the **Properties** tab.
3. Expand the **Attributes Grid** node, click the property to edit, click the **Conditional Updates** tab, and then do one of the following:
   - To configure a filtered condition, click **Condition Filter**.
   - To configure a scriptable condition, click **Script**.
4. Click **Save**.

Scriptable conditions create mapping logic, based on the result of the condition script. If the script does not return `true`, IDM does not manipulate the target attribute during a synchronization operation.

In the following excerpt, the value of the target `mail` attribute is set to the value of the source `email` attribute *only if* the source attribute is not empty:

```
{
    "target": "mail",
        "comment": "Set mail if non-empty.",
        "source": "email",
        "condition": {
            "type": "text/javascript",
            "source": "(object.email != null)"
        }
...
```

Tip

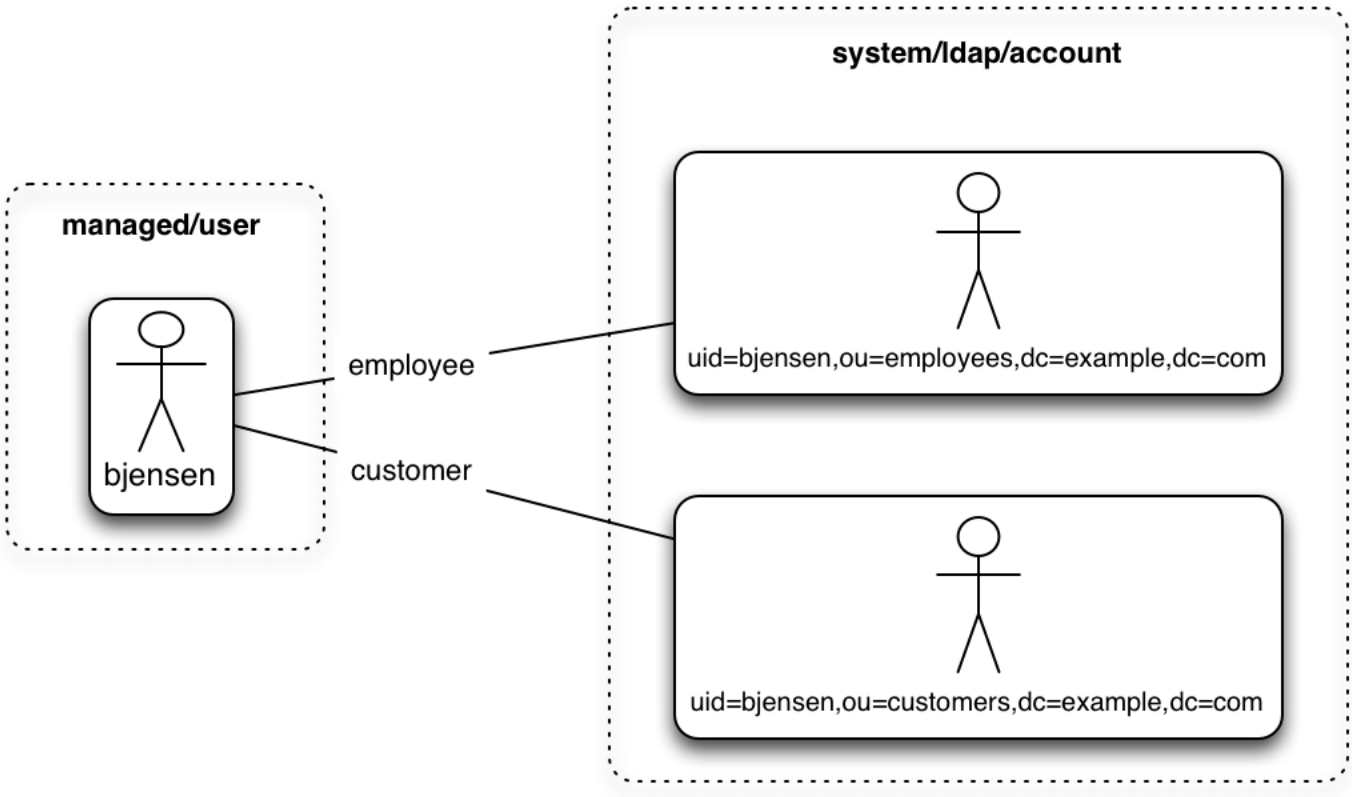
You can add comments to JSON files. This example includes a property named `comment`; however, you can use any unique property name, as long as it is not used elsewhere in the server. IDM ignores unknown property names in JSON configuration files.

Map a single source object to multiple target objects

In certain cases, you might have a single object in a resource that maps to more than one object in another resource. For example, assume that managed user, bjensen, has two distinct accounts in an LDAP directory: an `employee` account (under `uid=bjensen,ou=employees,dc=example,dc=com`) and a `customer` account (under `uid=bjensen,ou=customers,dc=example,dc=com`). You want to map both of these LDAP accounts to the same managed user account.

IDM uses *link qualifiers* to manage this one-to-many scenario. A link qualifier is essentially a label that identifies the *type* of link (or relationship) between objects.

The following diagram shows two link qualifiers that let you link both of bjensen's LDAP accounts to her managed user object:



Note

The previous diagram displays that the link qualifier is a property of the *link* between the source and target object, and not a property of the source or target object itself.

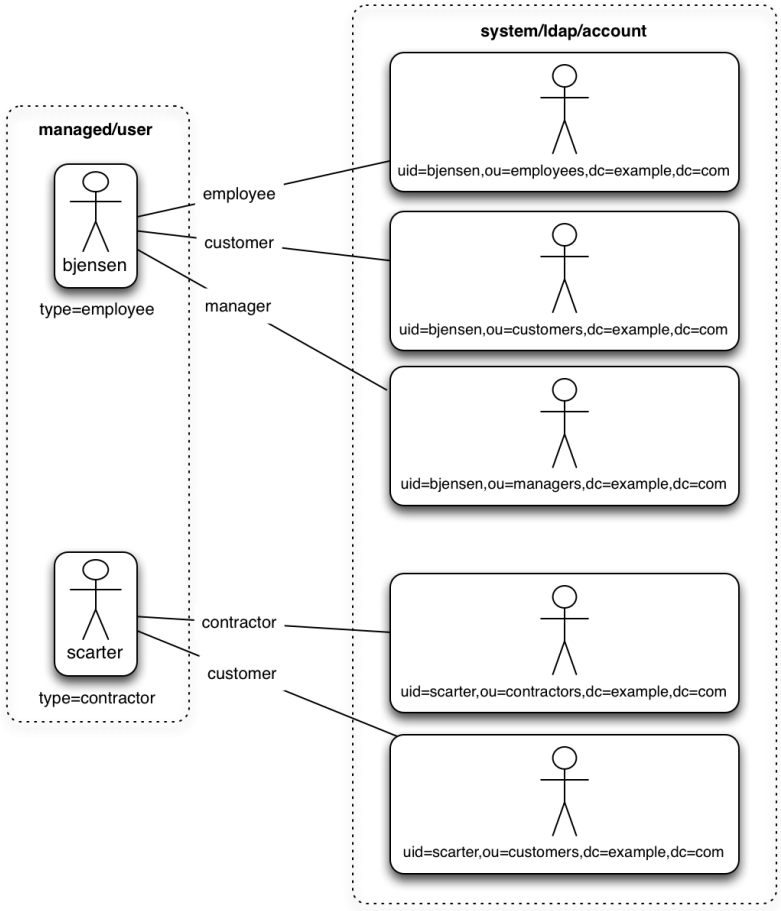
Link qualifiers are defined as part of the mapping. Each link qualifier must be unique within the mapping. If no link qualifier is specified (when only one possible matching target object exists), IDM uses a default link qualifier with the value `default`.

Link qualifiers can be defined as a static list, or dynamically, using a script. The following excerpt of a sample mapping shows the two static link qualifiers, `employee` and `customer`, described at the top of this topic:

```
{
    "mappings": [
        {
            "name": "managedUser_systemLdapAccounts",
            "source": "managed/user",
            "target": "system/MyLDAP/account",
            "linkQualifiers" : [ "employee", "customer" ],
...
```

IDM evaluates the list of static link qualifiers for *every* source record. That is, every reconciliation processes all synchronization operations, for each link qualifier, in turn.

A dynamic link qualifier script returns a list of link qualifiers that can be applied to each source record. For example, suppose you have two *types* of managed users—employees and contractors. For employees, a single managed user (source) account can correlate with three different LDAP (target) accounts—employee, customer, and manager. For contractors, a single managed user account can correlate with only two separate LDAP accounts—contractor, and customer. The following diagram displays the possible linking situations for this scenario:



In this scenario, you could write a script to generate a dynamic list of link qualifiers, based on the managed user type. For employees, the script would return `[employee, customer, manager]` in its list of possible link qualifiers. For contractors, the script would return `[contractor, customer]` in its list of possible link qualifiers. A reconciliation operation would then process only the list of link qualifiers applicable to each source object.

If your source resource includes many records, you should use a dynamic link qualifier script instead of a static list of link qualifiers. Generating the list of applicable link qualifiers dynamically avoids unnecessary additional processing for those qualifiers that will never apply to specific source records. Therefore, synchronization performance is improved for large source data sets.

You can include a dynamic link qualifier script inline (using the `source` property), or by referencing a JavaScript or Groovy script file (using the `file` property). The following link qualifier script sets up the dynamic link qualifier lists described in the previous example.

> **Note**
>
> In this example, the `source` property value has been formatted across multiple lines for clarity. In general, the script source must be formatted on a single line.

```
{
  "mappings": [
    {
      "name": "managedUser_systemLdapAccounts",
      "source": "managed/user",
      "target": "system/MyLDAP/account",
      "linkQualifiers" : {
        "type" : "text/javascript",
        "globals" : { },
        "source" : "if (returnAll) {
                      ['contractor', 'employee', 'customer', 'manager']
                    } else {
                      if(object.type === 'employee') {
                        ['employee', 'customer', 'manager']
                      } else {
                        ['contractor', 'customer']
                      }
                    }"
      }
...
```

To reference an external link qualifier script, provide a link to the file in the `file` property:

```
{
    "mappings": [
        {
            "name": "managedUser_systemLdapAccounts",
            "source": "managed/user",
            "target": "system/MyLDAP/account",
            "linkQualifiers" : {
                "type" : "text/javascript",
                "file" : "script/linkQualifiers.js"
            }
...
```

Dynamic link qualifier scripts must return all valid link qualifiers when the `returnAll` global variable is true. The `returnAll` variable is used during the target reconciliation phase to check whether there are any target records that are unassigned, for each known link qualifier.

If you configure dynamic link qualifiers through the UI, the complete list of dynamic link qualifiers displays in the Generated Link Qualifiers item below the script. This list represents the values returned by the script when the `returnAll` variable is passed as `true`. For a list of the variables available to a dynamic link qualifier script, refer to Script Triggers Defined in Mappings.

Link qualifiers have no functionality on their own, but they can be referenced in reconciliation operations to manage situations where a single source object maps to multiple target objects. The following examples show how link qualifiers can be used in reconciliation operations:

- Use link qualifiers during object creation, to create multiple target objects per source object.

The following mapping excerpt defines a transformation script that generates the value of the `dn` attribute on an LDAP system. If the link qualifier is `employee`, the value of the target `dn` is set to `"uid=userName,ou=employees,dc=example,dc=com"`. If the link qualifier is `customer`, the value of the target `dn` is set to `"uid=userName,ou=customers,dc=example,dc=com"`. The reconciliation operation iterates through the link qualifiers for each source record. In this case, two LDAP objects, with different `dn`s are created for each managed user object:

```
{
    "target" : "dn",
    "transform" : {
        "type" : "text/javascript",
        "globals" : { },
        "source" : "if (linkQualifier === 'employee')
              { 'uid=' + source.userName + ',ou=employees,dc=example,dc=com'; }
            else
            if (linkQualifier === 'customer')
              { 'uid=' + source.userName + ',ou=customers,dc=example,dc=com'; }"
    },
    "source" : ""
}
```

- Use link qualifiers with *correlation queries*. The correlation query assigns a link qualifier based on the values of an existing target object.

  During source synchronization, IDM queries the target system for every source record *and* link qualifier, to check if there are any matching target records. If a match is found, the sourceId, targetId, and linkQualifier are all saved as the *link*.

  The following excerpt of a sample mapping shows the two link qualifiers described previously (`employee` and `customer`). The correlation query first searches the target system for the `employee` link qualifier. If a target object matches the query, based on the value of its `dn` attribute, IDM creates a link between the source object and that target object, and assigns the `employee` link qualifier to that link. This process is repeated for all source records. Then, the correlation query searches the target system for the `customer` link qualifier. If a target object matches that query, IDM creates a link between the source object and that target object and assigns the `customer` link qualifier to that link:

  ```
  "linkQualifiers" : ["employee", "customer"],
    "correlationQuery" : [
      {
        "linkQualifier" : "employee",
        "type" : "text/javascript",
        "source" : "var query = {'_queryFilter': 'dn co \"' + uid=source.userName + 'ou=employees\"'}; query;"
      },
      {
        "linkQualifier" : "customer",
        "type" : "text/javascript",
        "source" : "var query = {'_queryFilter': 'dn co \"' + uid=source.userName + 'ou=customers\"'}; query;"
      }
    ]
  ...
  ```

  For more information about correlation queries, refer to Writing Correlation Queries.

- Use link qualifiers during policy validation to apply different policies based on the link type.

The following excerpt of a sample mapping shows two link qualifiers, `user` and `test`. Depending on the link qualifier, different actions are taken when the target record is ABSENT:

```
{
    "mappings" : [
        {
            "name" : "systemLdapAccounts_managedUser",
            "source" : "system/ldap/account",
            "target" : "managed/user",
            "linkQualifiers" : [
                "user",
                "test"
        ],
    "properties" : [
    ...
    "policies" : [
        {
            "situation" : "CONFIRMED",
            "action" : "IGNORE"
        },
        {
            "situation" : "FOUND",
            "action" : "UPDATE
        }
        {
            "condition" : "/linkQualifier eq \"user\"",
            "situation" : "ABSENT",
            "action" : "CREATE",
            "postAction" : {
                "type" : "text/javascript",
                "source" : "java.lang.System.out.println('Created user: \');"
            }
        },
        {
            "condition" : "/linkQualifier eq \"test\"",
            "situation" : "ABSENT",
            "action" : "IGNORE",
            "postAction" : {
                "type" : "text/javascript",
                "source" : "java.lang.System.out.println('Ignored user: ');"
            }
        },
        ...
```

With this sample mapping, the synchronization operation creates an object in the target system only if the potential match is assigned a `user` link qualifier. If the match is assigned a `test` qualifier, no target object is created. In this way, the process avoids creating duplicate *test-related* accounts in the target system.

Configure link qualifiers using the admin UI

1. From the navigation bar, click **Configure > Mappings**, and click the mapping to edit.
2. Click the **Properties** tab, and expand the **Link Qualifier** node.
3. Select **Static** or **Dynamic**, configure the link qualifier, and click **Save**.

> **Tip**
>
> For an example that uses link qualifiers in conjunction with roles, refer to Link Multiple Accounts to a Single Identity.

Prevent the accidental deletion of a target system

If a source resource is empty, the default behavior is to exit without failure and to log a warning similar to the following:

```
[318] Feb 19, 2020 1:51:56.455 PM org.forgerock.openidm.sync.NonClusteredRecon dispatchRecon
WARNING: Cannot reconcile from an empty data source, unless allowEmptySourceSet is true.
```

The reconciliation summary is also logged in the reconciliation audit log.

This behavior prevents reconciliation operations from accidentally deleting everything in a target resource. In the event that a source system is unavailable but erroneously reports its status as up, the absence of source objects should not result in objects being removed on the target resource.

If you *do* want reconciliations of an empty source resource to proceed, override the default behavior by setting the `allowEmptySourceSet` property to `true` in the mapping. For example:

```
{
    "mappings" : [
        {
        "name" : "systemCsvfileAccounts_managedUser",
        "source" : "system/csvfile/account",
        "allowEmptySourceSet" : true,
        ...
```

When an empty source is reconciled, the data in the target is wiped out.

Prevent accidental target deletion using the admin UI

1. From the navigation bar, click **Configure > Mappings**, and click the mapping to edit.
2. Click the **Advanced** tab, and expand the **Additional Mapping Options** node.
3. Enable **Allow Reconciliations From an Empty Source**.

Scripts in mappings

You can use a number of *script hooks* to manipulate objects and attributes during synchronization. Scripts can be triggered during various stages of the synchronization process, and are defined as part of the mapping.

You can trigger a script when a managed or system object is created (`onCreate`), updated (`onUpdate`), or deleted (`onDelete`). You can also trigger a script when a link is created (`onLink`) or removed (`onUnlink`).

In the default synchronization mapping, changes are always written to *target* objects, not to *source* objects. However, you can explicitly include a call to an action that should be taken on the source object within the script.

Construct and manipulate attributes

The most common use of synchronization scripts is when a target object is created or updated.

The `onUpdate` script is *always* called for an UPDATE situation, even if the synchronization process determines that there is no difference between the source and target objects, and that the target object will not be updated.

If the `onUpdate` script has run and the synchronization process then determines that the target value to set is the same as its existing value, the change is prevented from synchronizing to the target.

The following excerpt of a sample mapping derives a DN for an LDAP entry when the corresponding managed entry is created:

```
{
    "onCreate": {
        "type": "text/javascript",
        "source":
            "target.dn = 'uid=' + source.uid + ',ou=people,dc=example,dc=com'"
    }
}
```

Perform other actions

Construct and Manipulate Attributes With Scripts shows how to manipulate attributes with scripts when objects are created and updated. You can also trigger scripts in response to other synchronization actions. For example, you might not want 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 IDM and run a script to perform some subsequent action.

The following example shows a more advanced mapping configuration that exposes the script hooks available during synchronization:

```
{
    "mappings": [
        {
            "name": "systemLdapAccount_managedUser",
            "source": "system/ldap/account",
            "target": "managed/user",
            "validSource": {
                "type": "text/javascript",
                "file": "script/isValid.js"
            },
            "correlationQuery" : {
                "type" : "text/javascript",
                "source" : "var map = {'_queryFilter': 'uid eq \"' +
                     source.userName + '\"'}; map;"
            },
            "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"
                },
                {
                    "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",
                "source": "target.ldapPassword = null;
                    target.adPassword = null;
                    target.password = null;
                    target.ldapStatus = 'New Account'"
            },
            "onUpdate": {
                "type": "text/javascript",
                "source": "target.ldapStatus = 'OLD'"
            },
            "onUnlink": {
                "type": "text/javascript",
                "file": "script/triggerAdDisable.js"
            },
            "policies": [
                {
                    "situation": "CONFIRMED",
                    "action": "UPDATE"
                },
                {
                    "situation": "FOUND",
                    "action": "UPDATE"
                },
                {
                    "situation": "ABSENT",
                    "action": "CREATE"
                },
                {
                    "situation": "AMBIGUOUS",
                    "action": "EXCEPTION"
                },
                {
                    "situation": "MISSING",
                    "action": "EXCEPTION"
                },
                {
                    "situation": "UNQUALIFIED",
                    "action": "UNLINK"
                },
                {
                    "situation": "UNASSIGNED",
                    "action": "EXCEPTION"
                }
            ]
        }
    ]
}
```

The following list shows the properties that you can use as hooks in mapping configurations to call scripts:

Triggered by Situation

onCreate, onUpdate, onDelete, onLink, onUnlink

Object Filter

validSource, validTarget

Correlating Objects

correlationQuery

Triggered on Reconciliation

result

Scripts Inside Properties

condition, transform

Scripts can obtain data from any connected system 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/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb");
```

The following example reads an account from an external LDAP resource:

```
externalAccount = openidm.read("system/ldap/account/uid=bjensen,ou=People,dc=example,dc=com");
```

Important

For illustration purposes, this query targets a DN rather than a UID as it did in the previous example. The attribute that is used for the `_id` is defined in the connector configuration and, in this example, is set to `"uidAttribute" : "dn"`. Although you *can* use a DN (or any unique attribute) for the `_id`, it is a best practice to use an attribute that is both unique and immutable, such as the `entryUUID`.

Generate log messages

IDM provides a `logger` object that you can use from scripts defined in your mapping. These scripts can log messages to the OSGi console and to log files. The `logger` object includes the following functions:

- `debug()`
- `error()`
- `info()`
- `trace()`
- `warn()`

Consider the following mapping excerpt:

```
{
    "mappings" : [
        {
            "name" : "systemCsvfileAccounts_managedUser",
            "source" : "system/csvfile/account",
            "target" : "managed/user",
            "correlationQuery" : {
                "type" : "text/javascript",
                "source" : "var query = {'_queryId' : 'for-userName', 'uid' :  source.name};query;"
            },
            "onCreate" : {
                "type" : "text/javascript",
                "source" : "logger.warn('Case onCreate: the source object contains: = {} ', source); source;"
            },
            "onUpdate" : {
                "type" : "text/javascript",
                "source" : "logger.warn('Case onUpdate: the source object contains: = {} ', source); source;"
            },
            "result" : {
                "type" : "text/javascript",
                "source" : "logger.warn('Case result: the source object contains: = {} ', source); source;"
            },
            "properties" : [
                {
                    "transform" : {
                        "type" : "text/javascript",
                        "source" : "logger.warn('Case no Source: the source object contains: = {} ', source);
source;"
                    },
                    "target" : "sourceTest1Nosource"
                },
                {
                    "source" : "",
                    "transform" : {
                        "type" : "text/javascript",
                        "source" : "logger.warn('Case emptySource: the source object contains: = {} ', source);
source;"
                    },
                    "target" : "sourceTestEmptySource"
                },
                {
                    "source" : "description",
                    "transform" : {
                        "type" : "text/javascript",
                        "source" : "logger.warn('Case sourceDescription: the source object contains: = {} ', source);
source"
                    },
                    "target" : "sourceTestDescription"
                },
                ...
            ]
        }
    ]
}
```

The scripts that are defined for `onCreate`, `onUpdate`, and `result` log a warning message to the console whenever an object is created or updated, or when a result is returned. The script result includes the full source object.

The scripts that are defined in the `properties` section of the mapping log a warning message if the property in the source object is missing or empty. The last script logs a warning message that includes the description of the source object.

During a reconciliation operation, these scripts would generate output in the OSGi console, similar to the following:

```
2017-02... WARN Case no Source: the source object contains: = null  [9A00348661C6790E7881A7170F747F...]
2017-02... WARN Case emptySource: the source object contains: = {roles=openidm-..., lastname=Jensen...]
2017-02... WARN Case no Source: the source object contains: = null  [9A00348661C6790E7881A7170F747F...]
2017-02... WARN Case emptySource: the source object contains: = {roles=openidm..., lastname=Carter,...]
2017-02... WARN Case sourceDescription: the source object contains: = null  [EEE2FF4BCE9748927A1832...]
2017-02... WARN Case sourceDescription: the source object contains: = null  [EEE2FF4BCE9748927A1832...]
2017-02... WARN Case onCreate: the source object contains: = {roles=openidm-..., lastname=Carter,  ...]
2017-02... WARN Case onCreate: the source object contains: = {roles=openidm-..., lastname=Jensen,  ...]
2017-02... WARN Case result: the source object contains: = {SOURCE_IGNORED={count=0, ids=[]}, FOUND_ALL...]
```

You can use similar scripts to inject logging into any aspect of a mapping. You can also call the `logger` functions from any configuration file that has scripts hooks. For more information about the `logger` functions, refer to Log Functions.

Reuse links between mappings

When two mappings synchronize the same objects bidirectionally, use the `links` property in one mapping to have IDM use the same link for both mappings. If you do not specify a `links` property, IDM maintains a separate 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 indicates that IDM should 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"
        }
    ]
}
```

Reconcile with case-insensitive data stores

IDM is case-sensitive, which means that an uppercase ID is considered different from an otherwise identical lowercase ID during reconciliation. Some data stores, such as ForgeRock Directory Services (DS), are case-insensitive. This can be problematic during reconciliation, because the ID of the links created by reconciliation might not match the case of the IDs expected by IDM.

If a mapping inherits links by using the `links` property, you do not need to worry about case-sensitivity, because the mapping uses the setting of the referred links.

Alternatively, you can address case-sensitivity issues with target systems in the following ways:

- Specify a case-insensitive data store. To do so, set the `sourceIdsCaseSensitive` or `targetIdsCaseSensitive` properties to `false` in the mapping for those links. For example, if the source LDAP data store is case-insensitive, set the mapping from the LDAP store to the managed user repository as follows:

  ```
  "mappings" : [
      {
          "name" : "systemLdapAccounts_managedUser",
          "source" : "system/ldap/account",
          "sourceIdsCaseSensitive" : false,
          "target" : "managed/user",
          "properties" : [
      ...
  ```

  You might also need to modify the connector configuration, setting the `enableFilteredResultsHandler` property to `false`:

  ```
  "resultsHandlerConfig" :
  {
      "enableFilteredResultsHandler":false
  },
  ```

  Caution

  Do not disable the filtered results handler for the CSV file connector. The CSV file connector does not perform filtering. Therefore, if you disable the filtered results handler for this connector, the full CSV file will be returned for every request.

- Use a case-insensitive option in your managed repository. For example, for a MySQL repository, change the collation of `managedobjectproperties.propvalue` to `utf8_general_ci`. For more information, refer to Case insensitivity for a JDBC repo.

In general, to address case-sensitivity, focus on database-, table-, or column-level collation settings. Queries performed against repositories configured in this way are subject to the collation, and are used for comparison.

Synchronization situations and actions

The synchronization process assesses source and target objects, and the links between them, and then determines the *synchronization situation* that applies to each object. The process then performs a specific *action*, usually on the target object, depending on the assessed situation.

The action that is taken for each situation is defined in the `policies` section of your synchronization mapping.

The following excerpt of a sample mapping shows the defined actions in that sample:

```
{
    "policies": [
        {
            "situation": "CONFIRMED",
            "action": "UPDATE"
        },
        {
            "situation": "FOUND",
            "action": "LINK"
        },
        {
            "situation": "ABSENT",
            "action": "CREATE"
        },
        {
            "situation": "AMBIGUOUS",
            "action": "IGNORE"
        },
        {
            "situation": "MISSING",
            "action": "IGNORE"
        },
        {
            "situation": "SOURCE_MISSING",
            "action": "DELETE"
        },
        {
            "situation": "UNQUALIFIED",
            "action": "IGNORE"
        },
        {
            "situation": "UNASSIGNED",
            "action": "IGNORE"
        }
    ]
}
```

You can also define these actions using the admin UI:

1. From the navigation bar, click **Configure > Mappings**, and click the mapping to edit.
2. Click the **Behaviors** tab, expand the **Situational Event Scripts** node, and configure event actions.
3. Click **Save**.

> **Note**
>
> If you do not define an action for a particular situation, IDM takes the *default action* for that situation.

How IDM assesses synchronization situations

IDM performs reconciliation in two phases:

1. *Source reconciliation* accounts for source objects and associated links based on the configured mapping.
2. *Target reconciliation* iterates over the target objects that were not processed in the first phase.

For example, if a source object was deleted, the *source reconciliation* phase will not identify the target object that was previously linked to that source object. Instead, this *orphaned* target object is detected during the second phase.

Source reconciliation

During source reconciliation and liveSync, IDM iterates through the objects in the source resource. For reconciliation, the list of objects includes all objects that are available through the connector. For liveSync, the list contains only changed objects. IDM can filter objects from the list by using the following:

- Scripts specified in the `validSource` property
- A query specified in the `sourceCondition` property
- A query specified in the `sourceQuery` property

For each object in the list, IDM assesses the following conditions:

1. Is the source object valid?

   Valid source objects are categorized `qualifies=1`. Invalid source objects are categorized `qualifies=0`. Invalid objects include objects that were filtered out by a `validSource` script or `sourceCondition`. For more information, refer to Filter Source and Target Objects With Scripts.

2. Does the source object have a record in the links table?

   Source objects that have a corresponding link in the repository's `links` table are categorized `link=1`. Source objects that do not have a corresponding link are categorized `link=0`.

3. Does the source object have a corresponding valid target object?

   Source objects that have a corresponding object in the target resource are categorized `target=1`. Source objects that do not have a corresponding object in the target resource are categorized `target=0`.

The following diagram illustrates the categorization of four sample objects during source reconciliation. In this example, the source is the managed user repository and the target is an LDAP directory:

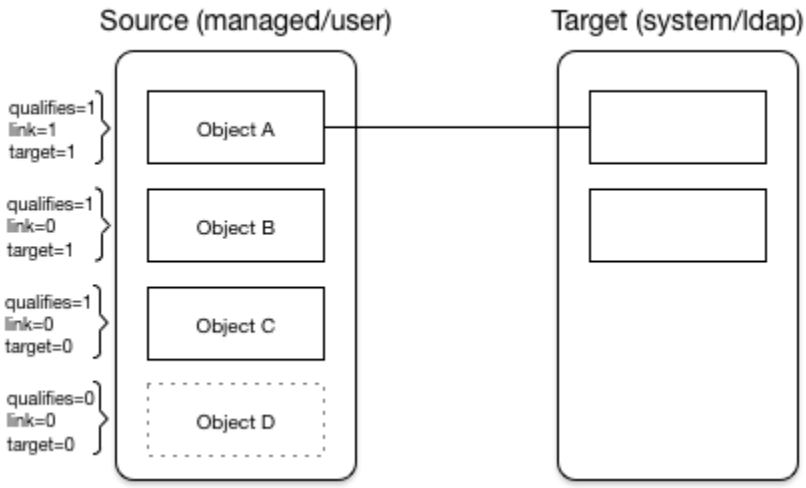


Figure 1. Object Categorization During the Source Synchronization Phase

Based on the categorizations of source objects during the source reconciliation phase, the synchronization process assesses a *situation* for each source object, and executes the *action* that is configured for each situation.

Not all situations are detected during all synchronization types (reconciliation, implicit synchronization, and liveSync). The following table describes the set of synchronization situations detected during source reconciliation, the default action taken for each situation, and valid alternative actions that can be configured for each situation:

Situations Detected During Reconciliation and Source Change Events

Source Qualifies	Link Exists	Target Objects Found	Situation	Default Action	Possible Actions
NO	**NO**	0	SOURCE_IGNORED	IGNORE source object	EXCEPTION, REPORT, NOREPORT, ASYNC
NO	**NO**	1	UNQUALIFIED	DELETE target object	EXCEPTION, IGNORE, REPORT, NOREPORT, ASYNC
NO	**NO**	> 1	UNQUALIFIED	DELETE target objects	EXCEPTION, IGNORE, REPORT, NOREPORT, ASYNC
NO	**YES**	0	UNQUALIFIED	DELETE linked target object [1]	EXCEPTION, REPORT, NOREPORT, ASYNC
NO	**YES**	1	UNQUALIFIED	DELETE linked target object	EXCEPTION, REPORT, NOREPORT, ASYNC
NO	**YES**	> 1	UNQUALIFIED	DELETE linked target object	EXCEPTION, REPORT, NOREPORT, ASYNC
YES	**NO**	0	ABSENT	CREATE target object	EXCEPTION, IGNORE, REPORT, NOREPORT, ASYNC
YES	**NO**	1	FOUND	UPDATE target object	EXCEPTION, IGNORE, REPORT, NOREPORT, ASYNC
YES	**NO**	1	FOUND_ALREADY_LINKED [2]	EXCEPTION	IGNORE, REPORT, NOREPORT, ASYNC
YES	**NO**	> 1	AMBIGUOUS [3]	EXCEPTION	REPORT, NOREPORT, ASYNC

Source Qualifies	Link Exists	Target Objects Found	Situation	Default Action	Possible Actions
YES	**YES**	0	MISSING [4]	EXCEPTION	CREATE, UNLINK, DELETE, IGNORE, REPORT, NOREPORT, ASYNC
YES	**YES**	1	CONFIRMED	UPDATE target object	IGNORE, REPORT, NOREPORT, ASYNC

Based on this table, the following situations would be assigned to the previous diagram:

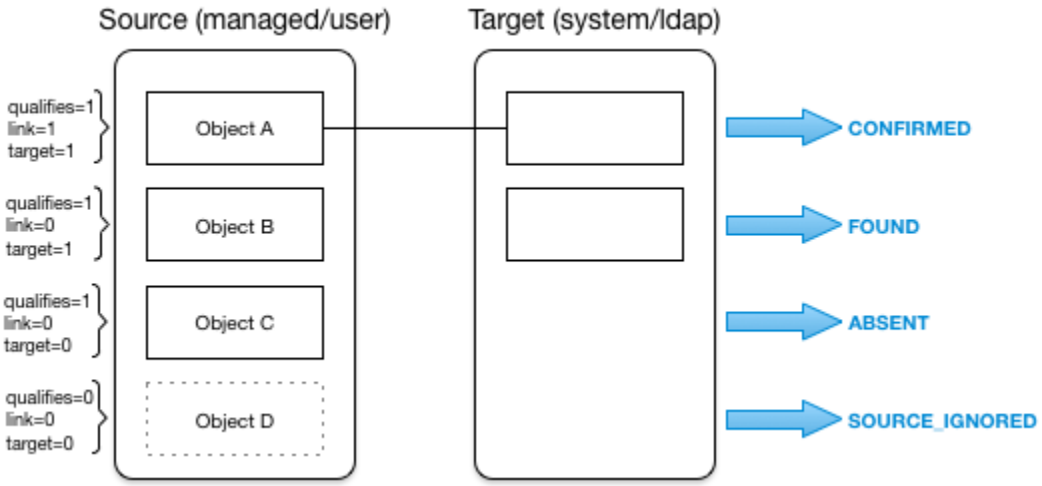


Figure 2. Situation Assignment During the Source Synchronization Phase

1. In this case (and the two following cases), the DELETE action is applied to the linked target object and not necessarily to the target object(s) found by the correlation query. If the source is no longer valid and a link existed, the correlation logic is skipped.
2. The source object qualifies for a target object and is not linked to an existing target object. There is a single target object that correlates with this source object, according to the logic in the correlation, but that target object is already linked to a different source object.
3. The source object qualifies for a target object, is not linked to an existing target object, but there is more than one correlated target object (that is, more than one possible match on the target system).
4. If the action is CREATE for the situation MISSING, the orphaned link associated with the source object is updated to point to the new target object. When a target object is deleted, the link from the target to the corresponding source object is not deleted automatically. This lets IDM detect and report items that might have been removed without permission or might need review. If you need to remove the corresponding link when a target object is deleted, change the action to UNLINK to remove the link, or to DELETE to remove the target object and the link.

Target reconciliation

During source reconciliation, the synchronization process cannot detect situations where no source object exists. In this case, the situation is detected during the second reconciliation phase, target reconciliation.

Target reconciliation iterates through the target objects that were not accounted for during source reconciliation. The process checks each object against the `validTarget` filter, determines the appropriate situation, and executes the action configured for the situation. Target reconciliation evaluates the following conditions:

1. Is the target object valid?

   Valid target objects are categorized `qualifies=1`. Invalid target objects are categorized `qualifies=0`. Invalid objects include objects that were filtered out by a `validTarget` script. For more information, refer to Filter Source and Target Objects With Scripts.

2. Does the target object have a record in the links table?

   Target objects that have a corresponding link in the `links` table are categorized `link=1`. Target objects that do not have a corresponding link are categorized `link=0`.

3. Does the target object have a corresponding source object?

   Target objects that have a corresponding object in the source resource are categorized `source=1`. Target objects that do not have a corresponding object in the source resource are categorized `source=0`.

The following diagram illustrates the categorization of three sample objects during target reconciliation:

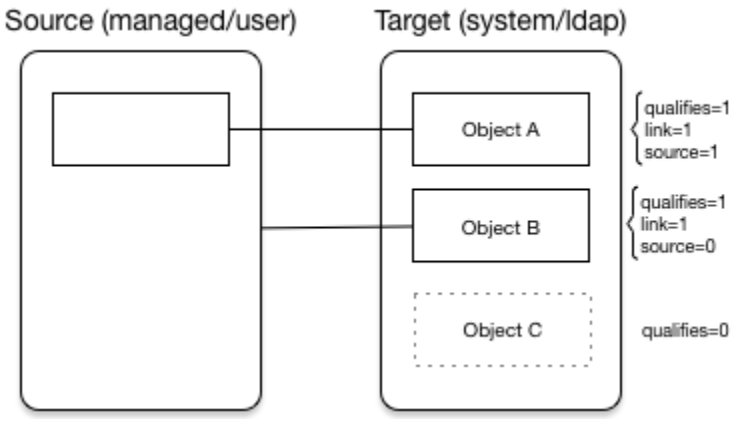


Figure 1. Object Categorization During the Target Synchronization Phase

Based on the categorizations of target objects during the target reconciliation phase, a *situation* is assessed for each remaining target object. Not all situations are detected in all synchronization types. The following table describes the set of situations that can be detected during the target reconciliation phase:

Situations Detected During Target Reconciliation

Target Qualifies	Link Exists	Source Exists	Source Qualifies	Situation	Default Action	Possible Actions
NO	n/a	n/a	n/a	TARGET_IGNORED [1]	IGNORE	DELETE, UNLINK, REPORT, NOREPORT, ASYNC
YES	**NO**	**NO**	n/a	UNASSIGNED	EXCEPTION	IGNORE, REPORT, NOREPORT, ASYNC

Target Qualifies	Link Exists	Source Exists	Source Qualifies	Situation	Default Action	Possible Actions
YES	**YES**	**YES**	**YES**	CONFIRMED	UPDATE target object	IGNORE, REPORT, NOREPORT
YES	**YES**	**YES**	**NO**	UNQUALIFIED [2]	DELETE	UNLINK, EXCEPTION, IGNORE, REPORT, NOREPORT, ASYNC
YES	**YES**	**NO**	n/a	SOURCE_MISSING [3]	EXCEPTION	DELETE, UNLINK, IGNORE, REPORT, NOREPORT, ASYNC

Based on this table, the following situations would be assigned to the previous diagram:

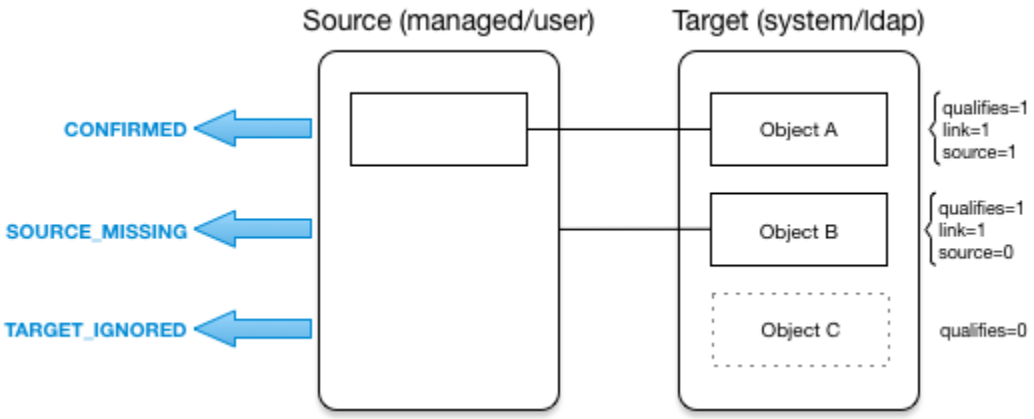


Figure 2. Situation Assignment During the Target Synchronization Phase

1. During target reconciliation, the target becomes unqualified by the `validTarget` script.
2. Detected during reconciliation and target change events
3. Detected during reconciliation and target change events

Situations specific to implicit synchronization and liveSync

Certain situations occur only during implicit synchronization (when changes made in the repository are pushed out to external systems) and liveSync (when IDM polls external system change logs for changes and updates the repository).

The following table shows the situations that pertain only to implicit sync and liveSync, when records are *deleted* from the source or target resource.

Situations detected during implicit sync or liveSync deletion events

Source Qualifies	Link Exists	Targets Found [1]	Targets Qualify	Situation	Default Action	Possible Actions
n/a	**YES**	0	n/a	LINK_ONLY	EXCEPTION	IGNORE, REPORT, NOREPORT, ASYNC

Source Qualifies	Link Exists	Targets Found [1]	Targets Qualify	Situation	Default Action	Possible Actions
n/a	**YES**	1	1	SOURCE_MISSING	EXCEPTION	IGNORE, REPORT, NOREPORT, ASYNC
n/a	**YES**	1	0	TARGET_IGNORED	IGNORE	DELETE, UNLINK, EXCEPTION, REPORT, NOREPORT, ASYNC
n/a	**NO**	0	n/a	ALL_GONE	IGNORE	EXCEPTION, REPORT, NOREPORT, ASYNC
YES	**NO**	0	n/a	ALL_GONE	IGNORE	EXCEPTION, REPORT, NOREPORT, ASYNC
YES	**NO**	1	1	UNASSIGNED	EXCEPTION	REPORT, NOREPORT
YES	**NO**	> 1	> 1	AMBIGUOUS	EXCEPTION	IGNORE, REPORT, NOREPORT, ASYNC
NO	**NO**	0	n/a	ALL_GONE	IGNORE	EXCEPTION, REPORT, NOREPORT, ASYNC
NO	**NO**	1	1	TARGET_IGNORED	IGNORE target object	DELETE, UNLINK, EXCEPTION, REPORT, NOREPORT, ASYNC
NO	**NO**	> 1	> 1	UNQUALIFIED	DELETE target objects	EXCEPTION, IGNORE, REPORT, NOREPORT, ASYNC

1. If no link exists for the source object, IDM executes any included correlation logic. If a link exists, correlation does not apply.

Synchronization actions

When an object has been assigned a situation, the synchronization process takes the configured *action* on that object. If no action is configured, the default action for that situation applies.

The following actions can be taken:

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.

> **Important**
>
> Do *not* use this action for liveSync mappings.
> In the context of liveSync, the EXCEPTION action triggers the liveSync failure handler, and the operation is retried in accordance with the configured retry policy. This is not useful because the operation will never succeed. If the configured number of retries is high, these pointless retries can continue for a long period of time.
> If the maximum number of retries is exceeded, the liveSync operation terminates and does not continue processing the entry that follows the failed (EXCEPTION) entry. LiveSync is only resumed at the next liveSync polling interval.
> This behavior differs from reconciliation, where a failure to synchronize a single source-target association does not fail the entire reconciliation.

IGNORE

Do not change the link or target object state.

REPORT

Do not perform any action but report what would happen if the default action were performed.

NOREPORT

Do not perform any action or generate any report.

ASYNC

An asynchronous process has been started, so do not perform any action or generate any report.

Launch a script as an action

In addition to the static synchronization actions described previously, you can provide a script to run in specific synchronization situations. The script can be either JavaScript or Groovy. You can specify the script inline (with the `"source"` property), or reference it from a file, (with the `"file"` property).

The following excerpt of a sample mapping specifies that an inline script should be invoked when a synchronization operation assesses an entry as `ABSENT` in the target system. The script checks whether the `employeeType` property of the corresponding source entry is `contractor`. If so, the source entry is ignored. Otherwise, the entry is created on the target system:

```
{
    "situation" : "ABSENT",
    "action" : {
        "type" : "text/javascript",
        "globals" : { },
        "source" : "if (source.employeeType === 'contractor') {action='IGNORE'}
                    else {action='CREATE'};action;"
    },
}
```

The following variables are available to a script that is called as an action:

- `source`
- `target`
- `linkQualifier`
- `recon` (where `recon.actionParam` contains information about the current reconciliation operation)

For more information about the variables available to scripts, refer to Script variables.

The result obtained from evaluating this script must be a string whose value is one of the synchronization actions listed in Synchronization actions. This resulting action is shown in the reconciliation log.

To launch a script as a synchronization action using the admin UI:

1. From the navigation bar, click **Configure > Mappings**, and click the mapping to edit.
2. Click the **Behaviors** tab, and expand the **Policies** node.
3. Click the edit button for the situation action to edit.
4. On the **Perform this Action** tab, click **Script**, and enter the script that corresponds to the action.
5. Click **Submit**, and then click **Save**.

Launch a workflow as an action

The `triggerWorkflowFromSync.js` script launches a predefined workflow when a synchronization operation assesses a particular situation. The mechanism for triggering this script is the same as for any other script. The script is provided in the `openidm/bin/defaults/script/workflow` directory. If you customize the script, copy it to the `script` directory of your project to ensure that your customizations are preserved during an upgrade.

The parameters for the workflow are passed as properties of the `action` parameter.

The following extract of a sample mapping specifies that, when a synchronization operation assesses an entry as `ABSENT`, the workflow named `managedUserApproval` is invoked:

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

To launch a workflow as a synchronization action Using the admin UI:

1. From the navigation bar, click **Configure > Mappings**, and click the mapping to edit.
2. Click the **Behaviors** tab, and expand the **Policies** node.
3. Click the edit button for the situation action to edit.
4. On the **Perform this Action** tab, click **Workflow**, and enter the details of the workflow to launch.
5. Click **Submit**, and then click **Save**.

Correlate source objects with existing target objects

When a synchronization operation creates an object on a target system, it also creates a *link* between the source and target object. IDM then uses that link to determine the object's *synchronization situation* during later synchronization operations. For a list of synchronization situations, refer to How IDM assesses synchronization situations.

Every synchronization operation can *correlate* existing source and target objects. Correlation matches source and target objects, based on the results of a query or script, and creates links between matched objects.

Correlation queries and correlation scripts are configured as part of the mapping. Each query or script is specific to the mapping for which it is configured.

Configure correlation using the admin UI

1. From the navigation bar, click **Configure > Mappings**.
2. From the **Mappings** page, click the mapping to correlate.
3. From the **Mapping Detail** page, click the **Association** tab.
4. Expand the **Association Rules** node, click the drop-down menu, and select one of the following:
    - **Correlation Queries**
    - **Correlation Script**
5. Build and/or write your script or query, and click **Save**.

Correlation queries

IDM processes a correlation query by constructing a query map. The content of the query is generated dynamically, using values from the source object. For each source object, a new query is sent to the target system, using (possibly transformed) values from the source object for its execution.

Queries are run against *target resources*, either managed or system objects, depending on the mapping. Correlation queries on system objects access the connector, which executes the query on the external resource.

You express a correlation query using a query filter (`_queryFilter`). For more information about query filters, refer to Define and call data queries. The synchronization process executes the correlation query to search through the target system for objects that match the current source object.

To configure a correlation query, define a script whose source returns a query that uses the `_queryFilter` , for example:

```
{ "_queryFilter" : "uid eq \"" + source.userName + "\"" }
```

Use filtered queries to correlate objects

For filtered queries, the script that is defined or referenced in the `correlationQuery` property must return an object with the following elements:

- The element that is being compared on the target object; for example, `uid` .

  The element on the target object is not necessarily a single attribute. Your query filter can be simple or complex; valid query filters range from a single operator to an entire boolean expression tree.

  If the target object is a system object, this attribute must be referred to by its IDM name rather than its OpenICF `nativeName` . For example, with the following provisioner configuration, the attribute to use in the correlation query would be `uid` and not `+NAME`` :

  ```
  ...
      "uid" : {
          "type" : "string",
          "nativeName" : "__NAME__",
          "required" : true,
          "nativeType" : "string"
      }
  ...
  ```

- The value to search for in the query.

  This value is generally based on one or more values from the source object. However, it does not have to match the value of a single source object property. You can define how your script uses the values from the source object to find a matching record in the target system.

  You might use a transformation of a source object property, such as `toUpperCase()` . You can concatenate that output with other strings or properties. You can also use this value to call an external REST endpoint, and redirect the response to the final "value" portion of the query.

The following correlation query matches source and target objects if the value of the `uid` attribute on the target is the same as the `userName` attribute on the source:

```
"correlationQuery" : {
    "type" : "text/javascript",
    "source" : "var qry = {'_queryFilter': 'uid eq \"' + source.userName + '\"'}; qry"
},
```

The query can return zero or more objects. The situation assigned to the source object depends on the number of target objects that are returned, and on the presence of any *link qualifiers* in the query. For information about synchronization situations, refer to How Synchronization Situations Are Assessed. For information about link qualifiers, refer to Map a Single Source Object to Multiple Target Objects.

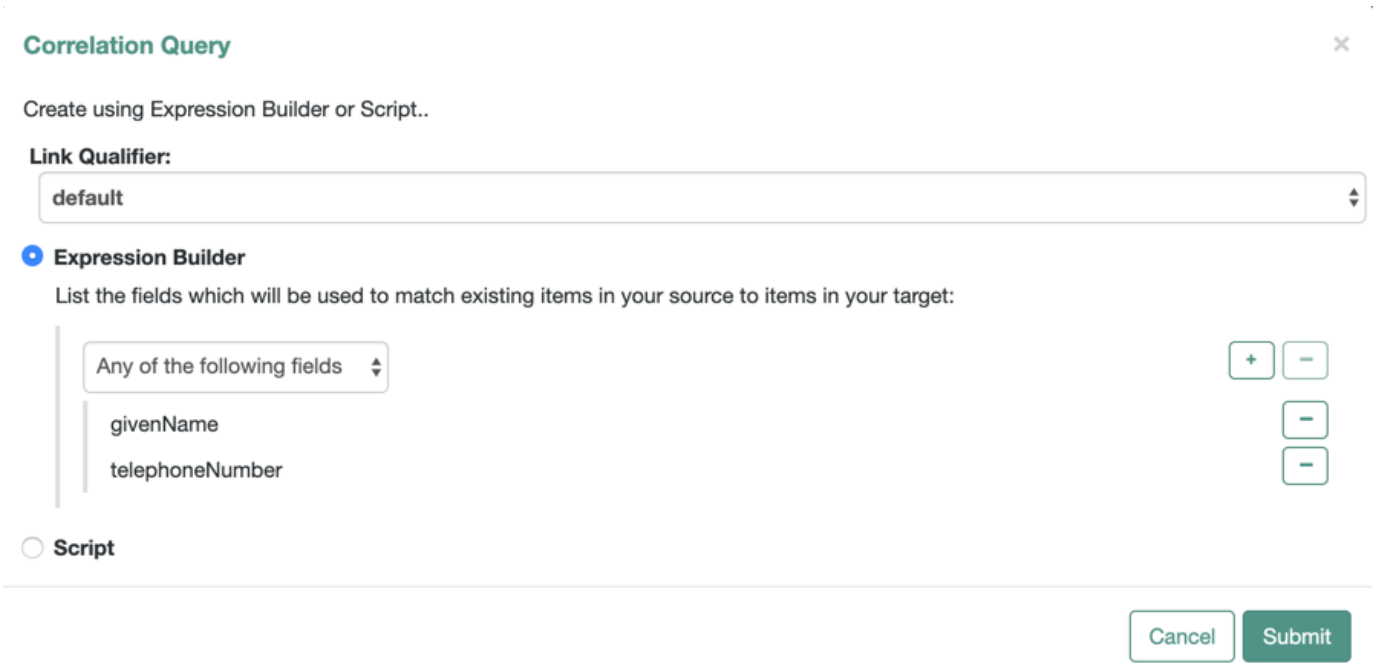
Create Correlation Queries Using the Expression Builder

The *Expression Builder* is a wizard that lets you quickly build expressions using drop-down menu options.

1. From the navigation bar, click **Configure > Mappings**.
2. On the **Mappings** page, click the mapping to correlate.
3. From the **Mapping Detail** page, click the **Association** tab.
4. Expand the **Association Rules** node, click the drop-down menu, and select **Correlation Queries**.
5. Click **Add Correlation Query**.
6. In the **Correlation Query** window, click the **Link Qualifier** drop-down menu, and select a link qualifier.

   If you do not need to correlate multiple potential target objects per source object, select the **default** link qualifier. For more information about linking to multiple target objects, refer to Map a Single Source Object to Multiple Target Objects.
7. Select **Expression Builder**.
8. To create an expression, use the drop-down menus to add and remove items, as necessary. List the fields to use for matching existing items in your source to items in your target.

   The following example displays an **Expression Builder** correlation query for a mapping from `managed/user` to `system/ldap/accounts` objects. The query creates a match between the source (managed) object and the target (LDAP) object if the value of the `givenName` or the `telephoneNumber` of those objects is the same.



9. After you finish building the expression, click **Submit**.
10. On the **Mapping Detail** page, under the **Association Rules** node, click **Save**.

The correlation query displays as follows in the mapping:

```
"correlationQuery" : [
    {
        "linkQualifier" : "default",
        "expressionTree" : {
            "any" : [
                "givenName",
                "telephoneNumber"
            ]
        },
        "mapping" : "managedUser_systemLdapAccounts",
        "type" : "text/javascript",
        "file" : "ui/correlateTreeToQueryFilter.js"
    }
]
```

Correlation scripts

In general, a correlation query should meet the requirements of most deployments. However, if you need a more powerful correlation mechanism than a simple query can provide, you can write a correlation script with additional logic. Correlation scripts can be useful if your query needs extra processing, such as fuzzy-logic matching or out-of-band verification with a third-party service over REST. Correlation scripts are generally more complex than correlation queries, and impose no restrictions on the methods used to find matching objects.

A correlation script must execute a query and return the result of that query. The result of a correlation script is a list of maps, each of which contains a candidate `_id` value. If no match is found, the script returns a zero-length list. If exactly one match is found, the script returns a single-element list. If there are multiple ambiguous matches, the script returns a list with multiple elements. There is no assumption that the matching target record or records can be found by a simple query on the target system. All of the work required to find matching records is left to the script.

To invoke a correlation script, use one of the following properties:

`correlationQuery`

Returns a `Map` whose values specify the `QueryFilter` for the sync engine to execute.

`correlationScript`

Returns a `List<Map>` whose value is a list of correlated objects from the target.

You can invoke a correlation script inline:

```
"correlationScript" : {
    "type": "text/javascript",
    "source": " var resultData = openidm.query("system/ldap/account", myQuery); return resultData.result;"
}
```

You can also invoke a correlation script using a script file:

```
"correlationScript" : {
    "type": "text/javascript",
    "file": "myCustomCorrelationScript.js"
}
```

Correlation Script Using Link Qualifiers

The following example shows a correlation script that uses link qualifiers. The script returns `resultData.result`—a list of maps, each of which has an `_id` entry. These entries will be the values that are used for correlation.

```
(function () {
    var query, resultData;
    switch (linkQualifier) {
        case "test":
            logger.info("linkQualifier = test");
                query = {'_queryFilter': 'uid eq \"' + source.userName + '-test\"'};
            break;
        case "user":
            logger.info("linkQualifier = user");
                query = {'_queryFilter': 'uid eq \"' + source.userName + '\"'};
            break;
        case "default":
            logger.info("linkQualifier = default");
                query = {'_queryFilter': 'uid eq \"' + source.userName + '\"'};
            break;
        default:
            logger.info("No linkQualifier provided.");
                break;
    }
    var resultData = openidm.query("system/ldap/account", query);
    logger.info("found " + resultData.result.length + " results for link qualifier " + linkQualifier)
    for (i=0;i<resultData.result.length;i++) {
        logger.info("found target: " + resultData.result[i]._id);
    }
    return resultData.result;
} ());
```

Configure a correlation script using the admin UI

1. From the navigation bar, click **Configure > Mappings**.
2. On the **Mappings** page, select the mapping to correlate.
3. From the **Mapping Detail** page, click the **Association** tab.
4. Expand the **Association Rules** node, click the drop-down menu, and select **Correlation Script**.
5. From the **Type** drop-down menu, select **JavaScript** or **Groovy**.
6. Enter the correlation script:
    - To use an inline script, select **Inline Script**, and type the script source.
    - To use a script file, select **File Path**, and enter the path to the script.

> **Tip**
>
> To create a correlation script, use the details from the source object to find the matching record in the target system. If you are using link qualifiers to match a single source record to multiple target records, you must also use the value of the `linkQualifier` variable within your correlation script to find the target ID that applies for that qualifier.

7. Click **Save**.

Synchronization operations

Manage reconciliation

To trigger, cancel, and monitor reconciliation operations over REST, use the `openidm/recon` REST endpoint. You can perform most of these actions using the admin UI.

Trigger a reconciliation

The following example triggers a reconciliation operation over REST based on the `systemLdapAccounts_managedUser` mapping:

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

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

The reconciliation run initiated previously would return something similar to the following:

```
{
  "_id": "05f63bce-4aaa-492e-9e86-a702d5c9d6c0-1144",
  "state": "ACTIVE"
}
```

To complete the reconciliation operation before the reconciliation run ID is returned, set the `waitForCompletion` property to `true` when the reconciliation is initiated:

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

> **Tip**
>
> To trigger this reconciliation using the admin UI, click **Configure > Mappings**, select a mapping, then click **Reconcile**. If you click **Cancel Reconciliation** before it completes, you will need to start the reconciliation again.

Cancel a reconciliation

To cancel an in progress reconciliation, specify the reconciliation run ID. The following REST call cancels the reconciliation run initiated in the previous section:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon/0890ad62-4738-4a3f-8b8e-f3c83bbf212e?_action=cancel"
```

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

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

If the reconciliation run is waiting for completion before its ID is returned, obtain the reconciliation run ID from the list of active reconciliations, as described in the following section.

> **Tip**
>
> To cancel a reconciliation run in progress using the admin UI, click **Configure > Mappings**, click on the mapping reconciliation to cancel, and click **Cancel Reconciliation**.

List reconciliation history

Display a list of reconciliation processes that have completed, and those that are in progress, by running a RESTful GET on `"http://localhost:8080/openidm/recon"`.

The following example displays all reconciliation runs:

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

The output is similar to the following, with one item for each reconciliation run:

```
"reconciliations": [
    {
      "_id": "05f63bce-4aaa-492e-9e86-a702d5c9d6c0-1144",
      "mapping": "systemLdapAccounts_managedUser",
      "state": "SUCCESS",
      "stage": "COMPLETED_SUCCESS",
      "stageDescription": "reconciliation completed.",
      "progress": {
        "source": {
          "existing": {
            "processed": 2,
            "total": "2"
          }
        },
        "target": {
          "existing": {
            "processed": 0,
            "total": "0"
          },
          "created": 2,
          "unchanged": 0,
          "updated": 0,
          "deleted": 0
        },
        "links": {
          "existing": {
            "processed": 0,
            "total": "0"
          },
          "created": 2
        }
      },
      "situationSummary": {
        "SOURCE_IGNORED": 0,
        "FOUND_ALREADY_LINKED": 0,
        "UNQUALIFIED": 0,
        "ABSENT": 2,
        "TARGET_IGNORED": 0,
        "MISSING": 0,
        "ALL_GONE": 0,
        "UNASSIGNED": 0,
        "AMBIGUOUS": 0,
        "CONFIRMED": 0,
        "LINK_ONLY": 0,
        "SOURCE_MISSING": 0,
        "FOUND": 0
      },
      "statusSummary": {
        "SUCCESS": 2,
        "FAILURE": 0
      },
      "durationSummary": {
        "sourceQuery": {
          "min": 42,
          "max": 42,
          "mean": 42,
          "count": 1,
          "sum": 42,
          "stdDev": 0
        },
```

```
          "auditLog": {
            "min": 0,
            "max": 1,
            "mean": 0,
            "count": 24,
            "sum": 15,
            "stdDev": 0
          },
          "linkQuery": {
            "min": 5,
            "max": 5,
            "mean": 5,
            "count": 1,
            "sum": 5,
            "stdDev": 0
          },
          "targetQuery": {
            "min": 3,
            "max": 3,
            "mean": 3,
            "count": 1,
            "sum": 3,
            "stdDev": 0
          },
          "targetPhase": {
            "min": 0,
            "max": 0,
            "mean": 0,
            "count": 1,
            "sum": 0,
            "stdDev": 0
          },
          "sourceObjectQuery": {
            "min": 6,
            "max": 34,
            "mean": 21,
            "count": 22,
            "sum": 474,
            "stdDev": 9
          },
          "postMappingScript": {
            "min": 0,
            "max": 1,
            "mean": 0,
            "count": 22,
            "sum": 17,
            "stdDev": 0
          },
          "onMappingScript": {
            "min": 0,
            "max": 4,
            "mean": 2,
            "count": 22,
            "sum": 48,
            "stdDev": 2
          },
          "sourcePhase": {
            "min": 490,
            "max": 490,
            "mean": 490,
            "count": 1,
            "sum": 490,
```

```
            "stdDev": 0
          }
        },
        "parameters": {
          "sourceQuery": {
            "resourceName": "system/ldap/account",
            "queryFilter": "true",
            "_fields": "_id"
          },
          "targetQuery": {
            "resourceName": "managed/user",
            "queryFilter": "true",
            "_fields": "_id"
          }
        },
        "started": "2020-05-07T09:14:57.740Z",
        "ended": "2020-05-07T09:14:58.325Z",
        "duration": 585,
        "sourceProcessedByNode": {}
      }
    ]
```

You can adjust the number of reconciliation runs that are stored in IDM by adding the `maxAnalysisRunsPerMapping` and `maxNonAnalysisRunsPerMapping` properties to your mapping:

```
"reconAssociation" : {
    "maxAnalysisRunsPerMapping" : 1,
    "maxNonAnalysisRunsPerMapping" : 3
}
```

In this context, *analysis* refers to reconciliation runs that are triggered with the `analyze=true` parameter. These runs don't perform any actions, but determine which actions *would* be performed in a real reconciliation. Non-analysis refers to a normal reconciliation. The default value for both properties is `1`.

In contrast, the admin UI displays the results of only the most recent reconciliation. For more information, refer to View reconciliation details using the admin UI.

Reconciliation Properties

Each reconciliation run includes the following properties:

`_id`

The ID of the reconciliation run.

`mapping`

The name of the mapping.

`state`

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

- `ACTIVE`

The reconciliation run is in progress.

- `CANCELED`

  The reconciliation run was successfully canceled.

- `FAILED`

  The reconciliation run was terminated because of failure.

- `SUCCESS`

  The reconciliation run completed successfully.

`stage`

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

- `ACTIVE_INITIALIZED`

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

- `ACTIVE_QUERY_ENTRIES`

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

- `ACTIVE_RECONCILING_SOURCE`

  Reconciling the set of IDs retrieved from the mapping source.

- `ACTIVE_RECONCILING_TARGET`

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

- `ACTIVE_LINK_CLEANUP`

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

- `ACTIVE_PROCESSING_RESULTS`

  Post-processing of reconciliation results.

- `ACTIVE_CANCELING`

  Attempting to abort a reconciliation run in progress.

- `COMPLETED_SUCCESS`

  Successfully completed processing the reconciliation run.

- `COMPLETED_CANCELED`

  Completed processing because the reconciliation run was aborted.

- `COMPLETED_FAILED`

  Completed processing because of a failure.

`stageDescription`

A description of the stages described previously.

`progress`

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

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

Reconciliation details

To display the details of a specific reconciliation over REST, include the reconciliation run ID in the URL. The following call shows the details of the reconciliation run initiated in Trigger a reconciliation.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/recon/05f63bce-4aaa-492e-9e86-a702d5c9d6c0-1144"
```

```
{
  "_id": "05f63bce-4aaa-492e-9e86-a702d5c9d6c0-1144",
  "mapping": "systemLdapAccounts_managedUser",
  "state": "SUCCESS",
  "stage": "COMPLETED_SUCCESS",
  "stageDescription": "reconciliation completed.",
  "progress": {
    "source": {
      "existing": {
        "processed": 2,
        "total": "2"
      }
    },
    "target": {
      "existing": {
        "processed": 0,
        "total": "0"
      },
      "created": 2,
      "unchanged": 0,
      "updated": 0,
      "deleted": 0
    },
    "links": {
      "existing": {
        "processed": 0,
        "total": "0"
      },
      "created": 2
    }
  },
  "situationSummary": {
    "SOURCE_IGNORED": 0,
    "FOUND_ALREADY_LINKED": 0,
    "UNQUALIFIED": 0,
    "ABSENT": 2,
    "TARGET_IGNORED": 0,
    "MISSING": 0,
    "ALL_GONE": 0,
    "UNASSIGNED": 0,
    "AMBIGUOUS": 0,
    "CONFIRMED": 0,
    "LINK_ONLY": 0,
    "SOURCE_MISSING": 0,
    "FOUND": 0
  },
  "statusSummary": {
    "SUCCESS": 2,
    "FAILURE": 0
  },
  "durationSummary": {
    "sourceQuery": {
      "min": 42,
      "max": 42,
      "mean": 42,
      "count": 1,
      "sum": 42,
      "stdDev": 0
    },
    "auditLog": {
```

```
        "min": 0,
        "max": 1,
        "mean": 0,
        "count": 24,
        "sum": 15,
        "stdDev": 0
      },
      "linkQuery": {
        "min": 5,
        "max": 5,
        "mean": 5,
        "count": 1,
        "sum": 5,
        "stdDev": 0
      },
      "targetQuery": {
        "min": 3,
        "max": 3,
        "mean": 3,
        "count": 1,
        "sum": 3,
        "stdDev": 0
      },
      "targetPhase": {
        "min": 0,
        "max": 0,
        "mean": 0,
        "count": 1,
        "sum": 0,
        "stdDev": 0
      },
      "sourceObjectQuery": {
        "min": 6,
        "max": 34,
        "mean": 21,
        "count": 22,
        "sum": 474,
        "stdDev": 9
      },
      "postMappingScript": {
        "min": 0,
        "max": 1,
        "mean": 0,
        "count": 22,
        "sum": 17,
        "stdDev": 0
      },
      "onMappingScript": {
        "min": 0,
        "max": 4,
        "mean": 2,
        "count": 22,
        "sum": 48,
        "stdDev": 2
      },
      "sourcePhase": {
        "min": 490,
        "max": 490,
        "mean": 490,
        "count": 1,
        "sum": 490,
        "stdDev": 0
```

```
      }
    },
    "parameters": {
      "sourceQuery": {
        "resourceName": "system/ldap/account",
        "queryFilter": "true",
        "_fields": "_id"
      },
      "targetQuery": {
        "resourceName": "managed/user",
        "queryFilter": "true",
        "_fields": "_id"
      }
    },
    "started": "2020-05-07T09:14:57.740Z",
    "ended": "2020-05-07T09:14:58.325Z",
    "duration": 585,
    "sourceProcessedByNode": {}
  }
```


View reconciliation details using the admin UI

You can display the details of the most recent reconciliation in the admin UI. Select the mapping. In the page that displays, you'll refer to a message similar to:

```
Completed: Last reconciled November 20, 2019 15:28
```

Clicking on the reconciliation run date displays the details of the reconciliation run. Click **Reconciliation Results** for additional information.

If a reconciliation fails, select the **Failure Summary** tab for more information about the failure.

To view reconciliation audit logs in the UI, add an Audit widget to your dashboard. The reconciliation Audit widget shows the same information that you get over REST.

Reconciliation association details

When performing a reconciliation run, information is reconciled between the source object and the target object. This creates an association between the two objects, which can be recorded in IDM by including the `persistAssociations=true` parameter when triggering a reconciliation. This information can then be retrieved by querying the `recon/assoc` endpoint.

To get a list of currently stored recon associations, run the following query:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/recon/assoc?_queryFilter=true"
{
  "result": [
    {
      "_id": "da88b9a5-1fe5-4f8d-a6a8-7e0a2b4e136b-9230",
      "_rev": "1",
      "mapping": "managedUser_systemLdapAccounts",
      "sourceResourceCollection": "managed/user",
      "targetResourceCollection": "system/ldap/account",
      "isAnalysis": "false",
      "finishTime": "2019-05-01T23:36:24.434153Z"
    },
    {
      "_id": "da88b9a5-1fe5-4f8d-a6a8-7e0a2b4e136b-99638",
      "_rev": "1",
      "mapping": "systemLdapAccounts_managedUser",
      "sourceResourceCollection": "system/ldap/account",
      "targetResourceCollection": "managed/user",
      "isAnalysis": "true",
      "finishTime": "2019-05-06T21:31:42.140066Z"
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

You can also get information for a specific reconciliation by querying the recon ID:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/recon/assoc/da88b9a5-1fe5-4f8d-a6a8-7e0a2b4e136b-9230"
{
  "_id": "da88b9a5-1fe5-4f8d-a6a8-7e0a2b4e136b-9230",
  "_rev": "1",
  "mapping": "managedUser_systemLdapAccounts",
  "sourceResourceCollection": "managed/user",
  "targetResourceCollection": "system/ldap/account",
  "isAnalysis": "false",
  "finishTime": "2019-05-01T23:36:24.434153Z"
}
```

It is possible to also get the specific association details of each entry in the reconciliation run by appending `/entry` to your query:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/recon/assoc/da88b9a5-1fe5-4f8d-a6a8-7e0a2b4e136b-9230/entry?
_queryFilter=true"
{
  "result": [
    {
      "_id": "400d40fd-da58-41f5-857b-71855eb97bd9",
      "_rev": "0",
      "mapping": "managedUser_systemLdapAccounts",
      "reconId": "da88b9a5-1fe5-4f8d-a6a8-7e0a2b4e136b-9230",
      "situation": "CONFIRMED",
      "action": "UPDATE",
      "linkQualifier": "default",
      "sourceObjectId": "07978ba5-b31d-4f8b-9f60-506c07f68495",
      "targetObjectId": "ca8abc7f-7b97-3e96-94fb-6b27b0ec5aed",
      "sourceResourceCollection": "managed/user",
      "targetResourceCollection": "system/ldap/account",
      "status": "SUCCESS",
      "exception": null,
      "message": null,
      "messageDetail": "null",
      "ambiguousTargetObjectIds": null
    },
    ...
    {
      "_id": "08ec633c-744f-4092-b88d-fe253b1d8e52",
      "_rev": "0",
      "mapping": "managedUser_systemLdapAccounts",
      "reconId": "da88b9a5-1fe5-4f8d-a6a8-7e0a2b4e136b-9230",
      "situation": "CONFIRMED",
      "action": "UPDATE",
      "linkQualifier": "default",
      "sourceObjectId": "ee2449a8-01e6-4c0b-84d3-e65e25c3e38c",
      "targetObjectId": "67a6596e-ebfc-3542-a664-1ab1610e082a",
      "sourceResourceCollection": "managed/user",
      "targetResourceCollection": "system/ldap/account",
      "status": "SUCCESS",
      "exception": null,
      "message": null,
      "messageDetail": "null",
      "ambiguousTargetObjectIds": null
    }
  ],
  ...
}
```

> **Note**
>
> For particularly large reconciliations, the results returned can be quite substantial, since it includes the details of every object reconciled. We encourage using query filters to tune your queries to only return the subset of results you're looking for.

Purge reconciliation statistics

When the number of completed reconciliation runs for a given mapping reaches the number specified by `maxAnalysisRunsPerMapping` or `maxNonAnalysisRunsPerMapping`, statistics are purged automatically. Statistics and reconciliation run information (such as recon associations) are purged chronologically by mapping, with the oldest reconciliation run for that mapping purged first.

You can also manually remove reconciliation statistics. To purge reconciliation statistics from the repository manually, run a DELETE command on the reconciliation run ID. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
"http://localhost:8080/openidm/recon/da88b9a5-1fe5-4f8d-a6a8-7e0a2b4e136b-9230"
```

Manage liveSync

Because you can trigger liveSync operations using REST (or the resource API) you can use an external scheduler to trigger liveSync operations, rather than using the IDM scheduling mechanism.

There are two ways to trigger liveSync over REST:

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

  ```
  curl \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --header "Accept-API-Version: resource=1.0" \
  --request POST \
  "http://localhost:8080/openidm/system/ldap/account?_action=liveSync"
  ```

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

  The following example calls the same liveSync operation as the previous example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=liveSync&source=system/ldap/account"
```

A successful liveSync operation returns the following response:

```
{
     "_rev": "000000001ade755f",
     "_id": "SYSTEMLDAPACCOUNT",
     "connectorData": {
         "nativeType": "JAVA_TYPE_LONG",
         "syncToken": 1
     }
}
```

Do not run two identical liveSync operations simultaneously. Rather, ensure that the first operation has completed before launching a second, similar operation.

Trigger liveSync using the admin UI

LiveSync operations are specific to a system object type (such as `system/ldap/account`). Apart from scheduling liveSync, as described in Scheduling LiveSync Through the UI, you can launch a liveSync operation on demand for a particular system object type as follows:

1. From the navigation bar, click **Configure** > **Connectors**.
2. On the **Connectors** page, select a connector.
3. On the **connector-name** page, click the **Object Types** tab.
4. Click the edit button adjacent to the object type to synchronize.
5. Click the **Sync** tab, and then click **Sync Now**.

   The **Sync Token** field displays the current synchronization token for the object type.

Troubleshoot liveSync failures

To troubleshoot a liveSync operation that has not succeeded, include the `detailedFailure` parameter to return additional information. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/ldap/account?_action=liveSync&detailedFailure=true"
```

The first time liveSync is called, it does not have a synchronization token in the database to establish which changes have already been processed. The default liveSync behavior is to locate the last existing entry in the change log, and to store that entry in the database as the current starting position from which changes should be applied. This behavior prevents liveSync from processing changes that might already have been processed during an initial data load. Subsequent liveSync operations will pick up and process any new changes.

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

In the case of DS, the change log (`cn=changelog`) can be read only by `uid=admin` by default. If you are configuring liveSync with DS, the `principal` that is defined in the LDAP connector configuration must have access to the change log. For information about allowing a regular user to read the change log, refer to Allow a User or Application to Read the Change Log.

If you refer to the following error message, you might have forgotten to set `changelog-read` access for a regular user:

```
Unable to locate the DS replication change log suffix. Please make
sure it's enabled, and changelog-read access is granted.
```

Filter synchronization data

By default, IDM synchronizes all objects that match those defined in the connector configuration for the resource. Many connectors let you limit the scope of objects that the connector accesses. For example, the LDAP connector lets you specify base DNs and LDAP filters so that you do not need to access every entry in the directory.

The following sections describe other ways to filter out objects or attributes to restrict the synchronization load.

Filter source and target objects with scripts

You can filter the source or target objects that are included in a synchronization operation using the `validSource` , `validTarget` , or `sourceCondition` properties in your mapping:

`validSource`

A script that determines if a source object is valid to be mapped.

The script yields a boolean value: `true` indicates that the source object is valid; `false` can be used to defer mapping until some condition is met. In the root scope, the source object is provided in the `"source"` property. If the script is not specified, then all source objects are considered valid:

```
{
    "validSource": {
        "type": "text/javascript",
        "source": "source.ldapPassword != null"
    }
}
```

`validTarget`

A script used during the second phase of reconciliation that determines if a target object is valid to be mapped.

The script yields a boolean value: `true` indicates that the target object is valid; `false` indicates that the target object should not be included in reconciliation. In the root scope, the source object is provided in the `"target"` property. If a `validTarget` the script is not specified, then all target objects are considered valid for mapping:

```
{
    "validTarget": {
        "type": "text/javascript",
        "source": "target.employeeType == 'internal'"
    }
}
```

`sourceCondition`

An additional filter that must be met for a source object to be included in a mapping.

This condition works like a `validSource` script. Its value can be either a `queryFilter` string, or a script configuration. `sourceCondition` is used mainly to specify that a mapping applies only to a particular role or entitlement.

The following `sourceCondition` restricts synchronization to those user objects whose account status is `active`:

```
{
    "mappings": [
        {
            "name": "managedUser_systemLdapAccounts",
            "source": "managed/user",
            "sourceCondition": "/source/accountStatus eq \"active\"",
        ...
        }
    ]
}
```

During synchronization, scripts and filters 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. Scripts can also write to target attributes using `target.attributeName` syntax, for example:

```
{
    "onUpdate": {
        "type": "text/javascript",
        "source": "if (source.email != null) {target.mail = source.email;}"
    }
}
```

The `sourceCondition` filter also has the `linkQualifier` variable in its scope.

For more information about scripting, refer to Scripting function reference.

Restrict reconciliation by using queries

Every reconciliation operation performs a query on the source and on the target resource, to determine which records should be reconciled. The default source and target queries are `_queryFilter=true&_fields=_id`, which means that all records in both the source and the target are considered candidates for that reconciliation operation.

You can restrict reconciliation to specific entries by defining an explicit `sourceQuery` or `targetQuery` in the mapping configuration.

> **Note**
>
> The `sourceQuery` filter is ignored during the target phase, and the `targetQuery` filter is ignored during the source phase.

For example, to restrict reconciliation to those records whose `employeeType` on the source resource is `Permanent`, you might specify a source query as follows:

```
"mappings" : [
     {
         "name" : "managedUser_systemLdapAccounts",
         "source" : "managed/user",
         "target" : "system/ldap/account",
         "sourceQuery" : {
            "_queryFilter" : "employeeType eq \"Permanent\""
         },
...
```

The format of the query can be any query type that is supported by the resource, and can include additional parameters, if applicable. Use the `_queryFilter` parameter, in common filter notation.

The source and target queries send the query to the resource that is defined for that source or target, by default. You can override the resource the query is sent to by specifying a `resourceName` in the query. For example, to query a specific endpoint instead of the source resource, you might modify the preceding source query as follows:

```
{
    "mappings" : [
        {
            "name" : "managedUser_systemLdapAccounts",
            "source" : "managed/user",
            "target" : "system/ldap/account",
            "sourceQuery" : {
                "resourceName" : "endpoint/scriptedQuery"
                "_queryFilter" : "employeeType eq \"Permanent\""
            },
            ...
}
```

To override a source or target query that is defined in the mapping, you can specify the query when you call the reconciliation operation. For example, to reconcile all employee entries, and not just the permanent employees, you would run the reconciliation operation as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{"sourceQuery": {"_queryFilter" : "true"}}' \
"http://localhost:8080/openidm/recon?_action=recon&mapping=managedUser_systemLdapAccounts"
```

By default, a reconciliation operation runs both the source and target phase. To avoid queries on the target resource, set `runTargetPhase` to `false` in the mapping configuration. To prevent the target resource from being queried during the reconciliation operation configured in the previous example, amend the mapping configuration as follows:

```
{
    "mappings" : [
        {
            "name" : "systemLdapAccounts_managedUser",
            "source" : "system/ldap/account",
            "target" : "managed/user",
            "sourceQuery" : {
                "_queryFilter" : "employeeType eq \"Permanent\""
            },
            "runTargetPhase" : false,
   ...
```

Restrict reconciliation queries using the admin UI

1. From the navigation bar, click **Configure > Mappings**.
2. On the **Mappings** page, select the mapping to restrict.
3. Click the **Association** tab, and expand the **Reconciliation Query Filters** node.
4. Create a source or target query, and click **Save**.

Restrict reconciliation to a specific ID

You can restrict reconciliation to a specific record in much the same way as you restrict reconciliation by using queries.

To restrict reconciliation to a specific ID, use the `reconById` action, instead of the `recon` action when you call the reconciliation operation. Specify the ID with the `id` parameter. Reconciling more than one ID with the `reconById` action is not supported.

The following command reconciles only the user with ID `b3c2f414-e7b3-46aa-8ce6-f4ab1e89288c`, for the mapping `managedUser_systemLdapAccounts`. The command synchronizes this particular user account in LDAP with the data from the managed user repository. The example assumes that implicit synchronization has been disabled, and that a reconciliation operation is required to copy changes made in the repository to the LDAP system:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?_action=reconById&mapping=managedUser_systemLdapAccounts&id=b3c2f414-e7b3-46aa-8ce6-f4ab1e89288c"
```

Reconciliation by ID takes the default reconciliation options that are specified in the mapping, so the source and target queries, and source and target phases apply equally to reconciliation by ID.

Restrict implicit synchronization to specific property changes

For a mapping that has managed objects as the source, an implicit synchronization is triggered if *any* source property changes, regardless of whether the modified property is explicitly defined as a `source` property in the mapping.

This default behavior is helpful in situations where no source properties are explicitly defined—any property within the object is included as part of the mapping.

However, this behavior adds a processing overhead, because every mapping from the managed object is invoked when *any* managed object property changes. If several mappings are configured from the managed object, this default behavior can cause performance issues.

In these situations, you can restrict the properties that should trigger an implicit synchronization *per mapping*, using the `triggerSyncProperties` attribute. This attribute contains an array of JSON pointers to the properties that must change before an implicit synchronization to the target is triggered. If none of these properties changes, no synchronization is triggered, even if other properties in the object change.

In the following example, implicit synchronization is triggered *only* if the `mail`, `telephoneNumber`, or `userName` of an object changes:

```
{
    "mappings" : [
        {
            "name" : "managedUser_systemLdapAccounts",
            "source" : "managed/user",
            "target" : "system/ldap/account",
            "enableLinking" : false,
            "triggerSyncProperties" : [
                "/mail",
                "/telephoneNumber",
                "/userName"
            ],
            "properties" : [],
            "policies" : []
        }
    ]
}
```

If any other property changes on the managed object, no implicit synchronization is triggered.

Implicit synchronization and liveSync

Implicit synchronization and *liveSync* refer to the automatic synchronization of changes from and to the managed object repository.

These topics describe the mechanisms for configuring these automatic synchronization mechanisms.

Array comparison

You can choose how synchronization detects managed object array changes using *unordered* or *ordered* comparison using the configuration property `comparison` in the schema. Unordered JSON array comparison ignores the order of elements and can negate the need for certain custom scripts within mappings.

Relationship and virtual property array fields default to unordered comparisons. All other fields default to ordered comparisons.

If you're using explicit mappings with a DS repository, you can't use ordered array comparison.

Learn more about managed object schema properties.

Disable automatic synchronization operations

By default, all mappings are automatically synchronized. A change to a managed object is automatically synchronized to all resources for which the managed object is configured as a source. If liveSync is enabled for a system, changes to an object on that system are automatically propagated to the managed object repository.

To prevent automatic synchronization for a specific mapping, set the `enableSync` property of that mapping to false. In the following example, implicit synchronization is disabled. This means that changes to objects in the internal repository are not automatically propagated to the LDAP directory. To propagate changes to the LDAP directory, reconciliation must be launched manually:

```
{
    "mappings" : [
        {
            "name" : "managedUser_systemLdapAccounts",
            "source" : "managed/user",
            "target" : "system/ldap/account",
            "enableSync" : false,
            ...
}
```

If `enableSync` is set to `false` for a mapping from a system resource to managed/user (for example `"systemLdapAccounts_managedUser"`), liveSync is disabled for that mapping.

Configure the liveSync retry policy

If a liveSync operation fails, IDM reattempts the change an infinite number of times until the change is successful. This behavior can increase data consistency in the case of transient failures (for example, when the connection to the database is temporarily lost). However, in situations where the cause of the failure is permanent (for example, if the change does not meet certain policy requirements) the change will never succeed, regardless of the number of attempts. In this case, the infinite retry behavior can effectively block subsequent liveSync operations from starting.

To avoid this, you can configure a liveSync retry policy to specify the number of times a failed modification should be reattempted, and what should happen if the modification is unsuccessful after the specified number of attempts.

Generally, a scheduled reconciliation operation will eventually force consistency. However, to prevent repeated retries that block liveSync, restrict the number of times that the same modification is attempted. You can then specify what happens to failed liveSync changes. The failed modification can be stored in a *dead letter queue* , discarded, or reapplied. Alternatively, an administrator can be notified of the failure by email or by some other means. This behavior can be scripted. The default configuration in the samples provided with IDM is to retry a failed modification five times, and then to log and ignore the failure.

You configure the liveSync retry policy in the connector configuration. The sample connector configurations have a retry policy defined as follows:

```
"syncFailureHandler" : {
  "maxRetries" : 5,
  "postRetryAction" : "logged-ignore"
},
```

`maxRetries`

Specifies the number of attempts that IDM should make to process the failed modification.

The value of this property must be a positive integer, or `-1` . A value of zero indicates that failed modifications should not be reattempted. In this case, the post-retry action is executed immediately when a liveSync operation fails. A value of `-1` (or omitting the `maxRetries` property, or the entire `syncFailureHandler` from the configuration) indicates that failed modifications should be retried an infinite number of times. In this case, no post retry action is executed.

The default retry policy relies on the scheduler, or whatever invokes liveSync. Therefore, if retries are enabled and a liveSync modification fails, IDM will retry the modification the next time that liveSync is invoked.

`postRetryAction`

Indicates what should happen if the maximum number of retries has been reached (or if `maxRetries` has been set to zero). The post-retry action can be one of the following:

- `logged-ignore`

  IDM should ignore the failed modification, and log its occurrence.

- `dead-letter-queue`

  IDM should save the details of the failed modification in a table in the repository (accessible over REST at `repo/synchronisation/deadLetterQueue/provisioner-name`).

- `script`

  Specifies a custom script that should be executed when the maximum number of retries has been reached. For information about using custom scripts in the configuration, refer to Scripting function reference. In addition to the regular objects described in that section, the following objects are available in the script scope:

  ##### `syncFailure`

  Provides details about the failed record. The structure of the `syncFailure` object is as follows:

```
"syncFailure" :
  {
    "token" : the ID of the token,
    "systemIdentifier" : a string identifier that matches the "name" property in the connector
configuration,
    "objectType" : the object type being synced, one of the keys in the "objectTypes" property
in the connector configuration,
    "uid" : the UID of the object (for example uid=joe,ou=People,dc=example,dc=com),
    "failedRecord", the record that failed to synchronize
  },
```

To access these fields, include `syncFailure.fieldname` in your script.

`failureCause`

Provides the exception that caused the original liveSync failure.

`failureHandlers`

Two synchronization failure handlers are provided by default:

- `loggedIgnore` indicates that the failure should be logged, after which no further action should be taken.
- `deadLetterQueue` indicates that the failed record should be written to a specific table in the repository, where further action can be taken.

> **Tip**
>
> To invoke one of the internal failure handlers from your script, use a call similar to the following (shown here for JavaScript):
>
> ```
> failureHandlers.deadLetterQueue.invoke(syncFailure, failureCause);
> ```

The following liveSync retry policy configuration specifies a maximum of four retries before the failed modification is sent to the dead letter queue:

```
...
    "syncFailureHandler" : {
        "maxRetries" : 4,
        "postRetryAction" : dead-letter-queue
    },
...
```

In the case of a failed modification, a message similar to the following is output to the logs:

```
INFO: sync retries = 1/4, retrying
```

IDM reattempts the modification the specified number of times. If the modification is still unsuccessful, a message similar to the following is logged:

```
INFO: sync retries = 4/4, retries exhausted
Jul 19, 2013 11:59:30 AM
    org.forgerock.openidm.provisioner.openicf.syncfailure.DeadLetterQueueHandler invoke
INFO: uid=jdoe,ou=people,dc=example,dc=com saved to dead letter queue
```

The log message indicates the entry for which the modification failed (`uid=jdoe` , in this example).

You can view the failed modification in the dead letter queue, over the REST interface, as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/repo/synchronisation/deadLetterQueue/ldap?_queryFilter=true&_fields=_id"
{
  "result":
    [
      {
        "_id": "4",
        "_rev": "000000001298f6a6"
      }
    ],
  ...
}
```

To view the details of a specific failed modification, include its ID in the URL:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/repo/synchronisation/deadLetterQueue/ldap/4"
{
  "objectType": "account",
  "systemIdentifier": "ldap",
  "failureCause": "org.forgerock.openidm.sync.SynchronizationException:
            org.forgerock.openidm.objset.ConflictException:
            org.forgerock.openidm.sync.SynchronizationException:
            org.forgerock.openidm.script.ScriptException:
            ReferenceError: \"bad\" is not defined.
            (PropertyMapping/mappings/0/properties/3/condition#1)",
  "token": 4,
  "failedRecord": "complete record, in xml format"
  "uid": "uid=jdoe,ou=people,dc=example,dc=com",
  "_rev": "000000001298f6a6",
  "_id": "4"
}
```

Note

The `repo` endpoint is an *internal interface*. Although it is used in the preceding example for the purposes of demonstration, you should not rely on this endpoint in production.

Improve reliability with queued synchronization

By default, IDM implicitly synchronizes managed object changes out to all resources for which the managed object is configured as a source. If there are several targets that must be synchronized, these targets are synchronized one at a time, one after the other. If any of the targets is remote or has a high latency, the implicit synchronization operations can take some time, delaying the successful return of the managed object change.

To decouple the managed object changes from the corresponding synchronizations, you can configure *queued synchronization*, which persists implicit synchronization events to the IDM repository. Queued events are then read from the repository and executed according to the queued synchronization configuration.

Because synchronization operations are performed in parallel, queued synchronization can improve performance if you have several fast, reliable targets. However, queued synchronization is also useful when your targets are slow or unreliable, because the managed object changes can complete before all targets have been synchronized.

The following illustration shows how synchronization operations are added to a local, in-memory queue. Note that this queue is distinct from the repository queue for synchronization events:

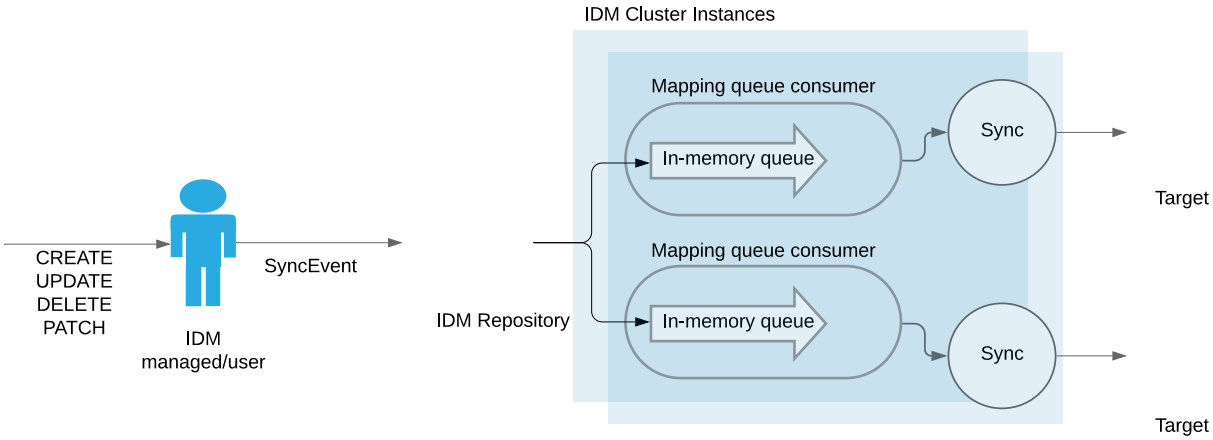


Figure 1. Queued Synchronization

Configure queued synchronization

Queued synchronization is disabled by default. To enable it, add a `queuedSync` object to your mapping, as follows:

```
{
     "mappings" : [
         {
             "name" : "managedUser_systemLdapAccounts",
             "source" : "managed/user",
             "target" : "system/ldap/account",
             "links" : "systemLdapAccounts_managedUser",
             "queuedSync" : {
                 "enabled" : true,
                 "pageSize" : 100,
                 "pollingInterval" : 1000,
                 "maxQueueSize" : 20000,
                 "maxRetries" : 5,
                 "retryDelay" : 1000,
                 "postRetryAction" : "logged-ignore"
             },
             ...
         }
     ]
 }
```

Note

- These settings apply *only* to the implicit synchronization operations for that mapping. Reconciliation is unaffected by queued synchronization settings. Events associated with mappings where queued synchronization is enabled are submitted to the synchronization queue for asynchronous processing. Events associated with mappings where queued synchronization is not enabled are processed immediately, and block further event processing until they are complete.
- During implicit synchronization, mappings are processed in the order in which they are defined, regardless of whether queued synchronization is enabled for those mappings. If you want all queued synchronization mappings to be processed first, you must explicitly order your mappings accordingly.
- Processing the synchronization queue for a mapping is *paused* if either the source or target system route is *unregistered*. A route is unregistered when you remove the connector configuration, set `"enabled" : false` in the connector configuration, delete the mapping, or remove the managed object type from the managed object configuration.

The `queuedSync` object has the following configuration:

`enabled`

Specifies whether queued synchronization is enabled for that mapping. Boolean, `true`, or `false`.

`pageSize` *(integer)*

Specifies the maximum number of events to retrieve from the repository queue within a single polling interval. The default is `100` events.

`pollingInterval` *(integer)*

Specifies the repository queue polling interval, in milliseconds. The default is `1000` ms.

`maxQueueSize` *(integer)*

Specifies the maximum number of synchronization events that can be accepted into the in-memory queue. The default is `20000` events.

`maxRetries` *(integer)*

The number of retries to perform before invoking the `postRetry` action. Most sample configurations set the maximum number of retries to `5`. To set an infinite number of retries, either omit the `maxRetries` property, or set it to a negative value, such as `-1`.

`retryDelay` *(integer)*

In the event of a failed queued synchronization operation, this parameter specifies the number of milliseconds to delay before attempting the operation again. The default is `1000` ms.

`postRetryAction`

The action to perform after the retries have been exhausted. Possible options are `logged-ignore`, `dead-letter-queue`, and `script`. These options are described in Configure the LiveSync Retry Policy. The default action is `logged-ignore`.

> **Note**
>
> Retries occur synchronously to the failure. For example, if the `maxRetries` is set to `10`, at least 10 seconds will pass between the failing sync event and the next sync. (There are 10 retries, and the `retryDelay` is 1 second by default.) These 10 seconds do not take into account the latency of the ten sync requests. Retries are configured per-mapping and block processing of all subsequent sync events until the configured retries have been exhausted.

Tune queued synchronization

Queued synchronization employs a single worker thread. While implicit synchronization operations are being generated, that worker thread should always be occupied. The occupation of the worker thread is a function of the `pageSize`, the `pollingInterval`, the latency of the poll request, and the latency of each synchronization operation for the mapping.

For example, assume that a poll takes 500 milliseconds to complete. Your system must provide operations to the worker thread at approximately the same rate at which the thread can consume events (based on the page size, poll frequency, and poll latency). Operation consumption is a function of the `notifyaction.execution` for that particular mapping. If the system does not provide operations fast enough, implicit synchronization will not occur as optimally as it could. If the system provides operations too quickly, the operations in the queue could exceed the default maximum of `20000`. If the `maxQueueSize` is reached, additional synchronization events will result in a `RejectedExecutionException`.

Depending on your hardware and workload, you might need to adjust the default `pageSize`, `pollingInterval`, and `maxQueueSize`.

Monitor the queued synchronization metrics; specifically, the `rejected-executions`, and adjust the `maxQueueSize` accordingly. Set a large enough `maxQueueSize` to prevent slow mappings and heavy loads from causing newly-submitted synchronization events to be rejected.

Monitor the synchronization latency using the `sync.queue.mapping-name.poll-pending-events` metric.

For more information on monitoring metrics, refer to Metrics reference.

Manage the synchronization queue

You can manage queued synchronization events over the REST interface, at the `openidm/sync/queue` endpoint. The following examples show the operations that are supported on this endpoint:

List all events in the synchronization queue:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/sync/queue?_queryFilter=true"
{
  "result": [
    {
      "_id": "03e6ab3b-9e5f-43ac-a7a7-a889c5556955",
      "_rev": "0000000034dba395",
      "mapping": "managedUser_systemLdapAccounts",
      "resourceId": "e6533cfe-81ad-4fe8-8104-55e17bd9a1a9",
      "syncAction": "notifyCreate",
      "state": "PENDING",
      "resourceCollection": "managed/user",
      "nodeId": null,
      "createDate": "2018-11-12T07:45:00.072Z"
    },
    {
      "_id": "ed940f4b-ce80-4a7f-9690-1ad33ad309e6",
      "_rev": "000000007878a376",
      "mapping": "managedUser_systemLdapAccounts",
      "resourceId": "28b1bd90-f647-4ba9-8722-b51319f68613",
      "syncAction": "notifyCreate",
      "state": "PENDING",
      "resourceCollection": "managed/user",
      "nodeId": null,
      "createDate": "2018-11-12T07:45:00.150Z"
    },
    {
      "_id": "f5af2eed-d83f-4b70-8001-8bc86075134f",
      "_rev": "00000000099aa321",
      "mapping": "managedUser_systemLdapAccounts",
      "resourceId": "d2691a45-0a10-4f51-aa2a-b6854b2f8086",
      "syncAction": "notifyCreate",
      "state": "PENDING",
      "resourceCollection": "managed/user",
      "nodeId": null,
      "createDate": "2018-11-12T07:45:00.276Z"
    },
    ...
  ],
  "resultCount": 8,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

Query the queued synchronization events based on the following properties:

- `mapping`—the mapping associated with this event. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/sync/queue?_queryFilter=mapping+eq+'managedUser_systemLdapAccount'"
```

- `nodeId` —the ID of the node that has acquired this event.
- `resourceId` —the source object resource ID.
- `resourceCollection` —the source object resource collection.
- `_id` —the ID of this sync event.
- `state` —the state of the synchronization event. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/sync/queue?_queryFilter=state+eq+'PENDING'"
```

  The `state` of a queued synchronization event is one of the following:

  - `PENDING` —the event is waiting to be processed.
  - `ACQUIRED` —the event is being processed by a node.

- `remainingRetries` —the number of retries available for this synchronization event before it is abandoned. For more information about how synchronization events are retried, refer to Configure the LiveSync Retry Policy. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/sync/queue?_queryFilter=remainingRetries+lt+2"
```

- `syncAction` —the synchronization action that initiated this event. Possible synchronization actions are `notifyCreate`, `notifyUpdate`, and `notifyDelete`. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/sync/queue?_queryFilter=syncAction+eq+'notifyCreate'"
```

- `createDate` —the date that the event was created.

Recover mappings when nodes are down

Synchronization events for mappings with queued synchronization enabled are processed by a single cluster node. While a node is present in the cluster, that node holds a *lock* on the specific mapping. The node can release or reacquire the mapping lock if a balancing event occurs (see Balance Mapping Locks Across Nodes). However, the mapping lock is held across all events on that mapping. In a stable running cluster, a single node will hold the lock for a mapping indefinitely.

It is possible that a node goes down, or is removed from the cluster, while holding a mapping lock on operations in the synchronization queue. To prevent these operations from being lost, the queued synchronization facility includes a *recovery monitor* that checks for any *orphaned* mappings in the cluster.

A mapping is considered orphaned in the following cases:

- No active node holds a lock on the mapping.
- The node that holds a lock on the mapping has an instance state of `STATE_DOWN`.
- The node that holds a lock on the mapping does not exist in the cluster.

The recovery monitor periodically checks for orphaned mappings. When all orphaned mappings have been recovered, it attempts to initialize new queue consumers.

The recovery monitor is enabled by default and executes every 300 seconds. To change the default behavior for a mapping, add the following to the mapping configuration and change the parameters as required:

```
{
    "mappings" : [...],
    "queueRecovery" : {
        "enabled" : true,
        "recoveryInterval" : 300
    }
}
```

Important

If a queued synchronization job has already been claimed by a node, and that node is *shut down*, IDM notifies the entire cluster of the shutdown. This lets a different node pick up the job in progress. The recovery monitor takes over jobs in a synchronization queue that have not been fully processed by an available cluster node, so no job should be lost. If you have configured queued synchronization for one or more mappings, do not use the `enabled` flag in the cluster configuration to remove a node from the cluster. Instead, shut down the node so that the remaining nodes in the cluster can take over the queued synchronization jobs.

Balance mapping locks across nodes

Queued synchronization mapping locks are balanced equitably across cluster nodes. At a specified interval, each node attempts to release and acquire mapping locks, based on the number of running cluster nodes. When new cluster nodes come online, existing nodes release sufficient mapping locks for new nodes to pick them up, resulting in an equitable distribution of locks.

Lock balancing is enabled by default, and the interval at which nodes attempt to balance locks in the queue is 5 seconds. To change the default configuration, add a `queueBalancing` object to your mapping and set the following parameters:

```
{
    "mappings" : [...],
    "queueBalancing" : {
        "enabled" : true,
        "balanceInterval" : 5
    }
}
```

Synchronization failure compensation

If implicit synchronization fails for a target resource (for example, due to a policy validation failure on the target, or the target being unavailable), the synchronization operation stops at that point. In this scenario, a record might be changed in the repository, and in the targets on which synchronization was successful, but not on the failed target, or on any targets that would have been synchronized *after* the failure. This can result in disparate data sets across resources. Although a reconciliation operation would eventually bring all targets back in sync, reconciliation can be an expensive operation with large data sets.

You can configure *synchronization failure compensation* to prevent data sets from becoming out of sync. This mechanism involves reverting an implicit synchronization operation if it is not completely successful across all configured mappings.

Failure compensation ensures that either all resources are synchronized successfully, or that the original change is rolled back. This mechanism uses an `onSync` script hook in the managed object configuration. The `onSync` hook calls a script that prevents partial synchronization by "reverting" a partial change in the event that all resources are not synchronized.

The following sample managed object configuration shows the addition of the `onSync` hook:

```
...
"onDelete" : {
    "type" : "text/javascript",
    "file" : "onDelete-user-cleanup.js"
},
"onSync" : {
    "type" : "text/javascript",
    "file" : "compensate.js"
},
"properties" : [
    ...
```

With this configuration, a change to a managed object triggers an implicit synchronization for each configured mapping, in the order in which the mappings are defined. If synchronization is successful for all configured mappings, IDM exits from the script. If synchronization fails for a particular resource, the `onSync` hook invokes the `compensate.js` script, which attempts to revert the original change by performing another update to the managed object. This change, in turn, triggers another implicit synchronization operation to all external resources for which mappings are configured.

If the synchronization operation fails again, the `compensate.js` script is triggered a second time. This time, however, the script recognizes that the change was originally called as a result of a compensation and aborts. IDM logs warning messages related to the sync action (`notifyCreate` , `notifyUpdate` , `notifyDelete`), along with the error that caused the sync failure.

If failure compensation is not configured, any issues with connections to an external resource can result in out of sync data stores.

With the `compensate.js` script, any such errors will result in each data store retaining the information it had before implicit synchronization started. That information is stored, temporarily, in the `oldObject` variable.

Schedule synchronization

You can schedule synchronization operations, such as liveSync and reconciliation, using Quartz triggers. IDM supports simple triggers and cron triggers.

Use the trigger type that suits your scheduling requirements. Because simple triggers are not bound to the local timezone, they are better suited to scenarios such as liveSync, where the requirement is to trigger the schedule at regular intervals, regardless of the local time. For more information, refer to the Quartz documentation on SimpleTriggers and CronTriggers.

This section describes scheduling specifically for reconciliation and liveSync, and shows simple triggers in all the examples. You can use the scheduler service to schedule any other event by supplying a script in which that event is defined. For information about scheduling other events, refer to Schedule tasks and events.

Configure scheduled synchronization

Each scheduled reconciliation and liveSync task requires a schedule configuration, with the following format:

```
{
 "enabled"        : boolean, true/false
 "type"           : "string",
 "repeatInterval" : long integer,
 "repeatCount"    : integer,
 "persisted"      : boolean, true/false
 "startTime"      : "(optional) time",
 "endTime"        : "(optional) time",
 "schedule"       : "cron expression",
 "misfirePolicy"  : "optional, string",
 "invokeService"  : "service identifier",
 "invokeContext"  : "service specific context info"
}
```

These properties are specific to the scheduler service, and are explained in Schedule tasks and events.

To schedule a reconciliation or liveSync task, set the `invokeService` property to either `sync` (for reconciliation) or `provisioner` for liveSync.

The value of the `invokeContext` property depends on the type of scheduled event. For reconciliation, the properties are set as follows:

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

The `mapping` is referenced by its `name` in the mapping configuration.

For liveSync, the properties are set as follows:

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

The `source` property follows the convention for a pointer to an external resource object, and takes the form `system/resource-name/object-type`.

> **Important**
>
> When you schedule a reconciliation operation to run at regular intervals, do not set `"concurrentExecution" : true`. This parameter enables multiple scheduled operations to run concurrently. You cannot launch multiple reconciliation operations for a single mapping concurrently.

Schedule liveSync using the admin UI

To configure liveSync using the admin UI, set up a liveSync schedule:

1. From the navigation bar, click **Configure > Schedules**, and then click **Add Schedule**.
2. Complete the schedule configuration, and click **Save**.

   For more information about these fields, refer to Configure Scheduled Synchronization.

> **Note**
>
> The scheduler configuration assumes a `simple` trigger type by default, so the `Cron-like Trigger` field is disabled. You should use simple triggers for liveSync schedules to avoid problems related to daylight savings time. For more information, refer to Schedules and daylight savings time.
> By default, the admin UI creates schedules using the scheduler service, rather than the configuration service. To create this schedule in the configuration service, select the **Save as Config Object** option.
> If your deployment enables writes to configuration files, this option also creates a corresponding `schedule-schedule-name.json` file in your project's `conf` directory.
> For more information on the distinction between the scheduler service and the configuration service, refer to Configure the scheduler service.

Clustered reconciliation

In a clustered deployment, you can configure reconciliation jobs to be distributed across multiple nodes in the cluster. Clustered reconciliation is configured *per mapping* and can improve reconciliation performance, particularly for very large data sets.

Clustered reconciliation uses the paged reconciliation mechanism and the scheduler service to divide the *source* data set into pages, and then to schedule reconciliation "sub-jobs" per page, distributing these sub-jobs across the nodes in the cluster.

Regular (non-clustered) reconciliation has two phases—a source phase and a target phase. Clustered reconciliation effectively has three phases:

Source page phase

During this phase, reconciliation sub-jobs are scheduled in succession, page by page. Each source page job does the following:

- Executes a source query using the paging cookie from the invocation context.
- Schedules the next source page job.
- Performs the reconciliation of the source IDs returned by the query.
- Writes statistics summary information which is aggregated so that you can obtain the status of the complete reconciliation run by performing a GET on the `recon` endpoint.
- On completion, writes the `repo_id`, `source_id`, and `target_id` to the repository.

Source phase completion check

This phase is scheduled when the source query returns null. This check runs, and continues to reschedule itself, as long as source page jobs are running. When the completion check determines that all the source page jobs are complete, it schedules the target phase.

Target phase

This phase queries the target IDs, then removes all of the IDs that correspond to the `repo_id`, `source_id`, and `target_id` written by the source pages. The remaining target IDs are used to run the target phase, taking into account all records on the target system that were not correlated to a source ID during the source phase sub-jobs.

Configure clustered reconciliation for a mapping

To specify that the reconciliation for a specific mapping should be distributed across a cluster, add the `clusteredSourceReconEnabled` property to the mapping and set it to `true`. For example:

```
{
    "mappings" : [
        {
            "name" : "systemLdapAccounts_managedUser",
            "source" : "system/ldap/account",
            "target" : "managed/user",
            "clusteredSourceReconEnabled" : true,
  ...
}
```

Note

When clustered reconciliation is enabled, source query paging is enabled automatically, regardless of the value that you set for the `reconSourceQueryPaging` property in the mapping.

By default, the number of records per page is **10000**. You can also enable *target query paging* with the `reconTargetQueryPaging` property (defaults to `false`). To change the query page sizes, set the `reconSourceQueryPageSize` and `reconTargetQueryPageSize` properties.

The following example enables *target query paging* and changes the target and source query page sizes:

```
{
      "mappings" : [
          {
              "name" : "systemLdapAccounts_managedUser",
              "source" : "system/ldap/account",
              "target" : "managed/user",
              "clusteredSourceReconEnabled" : true,
              "reconTargetQueryPaging" : true,
              "reconSourceQueryPageSize" : 12000,
              "reconTargetQueryPageSize" : 12000,
          ...
}
```

To set these properties using the admin UI, click **Configure > Mappings**, select the mapping to change, and click the **Advanced** tab.

Clustered reconciliation has the following limitations:

- A complete non-clustered reconciliation run is synchronous with the single reconciliation invocation.

  By contrast, a clustered reconciliation is *asynchronous*. In a clustered reconciliation, the first execution is synchronous only with the reconciliation of the first page. This job also schedules the subsequent pages of the clustered reconciliation to run on other cluster nodes. When you schedule a clustered reconciliation or call the operation over REST, do not set `waitForCompletion` to `true`, because you cannot wait for the operation to complete before the next operation starts.

  Because this first execution does not encompass the entire reconciliation operation for that mapping, you cannot rely on the Quartz `concurrentExecution` property to prevent two reconciliation operations from running concurrently. If you use Quartz to schedule clustered reconciliations (as described in Configure Scheduled Synchronization), make sure that the interval between scheduled operations exceeds the known run of the entire clustered reconciliation. The run-length of a specific clustered reconciliation can vary. You must therefore build in appropriate buffer times between schedules, or use a scheduled script that performs a GET on the `recon/` endpoint, and dispatches the next reconciliation on a mapping only when the previous reconciliation run has completed.

- Clustered reconciliations can recover missing source pages (for example, if a cluster goes offline during a clustered reconciliation run), except when used with a connector using server-side logic to handle paging that returns a static paging cookie.

Clustered reconciliation progress

The `sourceProcessedByNode` property indicates how many records are processed by each node. You can verify the load distribution per node by running a GET on the recon endpoint, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/recon"
...
    "started": "2017-05-11T10:04:59.563Z",
    "ended": "",
    "duration": 342237,
    "sourceProcessedByNode": {
        "node2": 21500,
        "node1": 22000
    }
}
```

You can also display the nodes responsible for each source page in the admin UI. Click on the relevant mapping and expand the In Progress or Reconciliation Results item. The following image shows a clustered reconciliation in progress. The details include the number of records that have been processed, the current duration of the reconciliation, and the load distribution, per node:

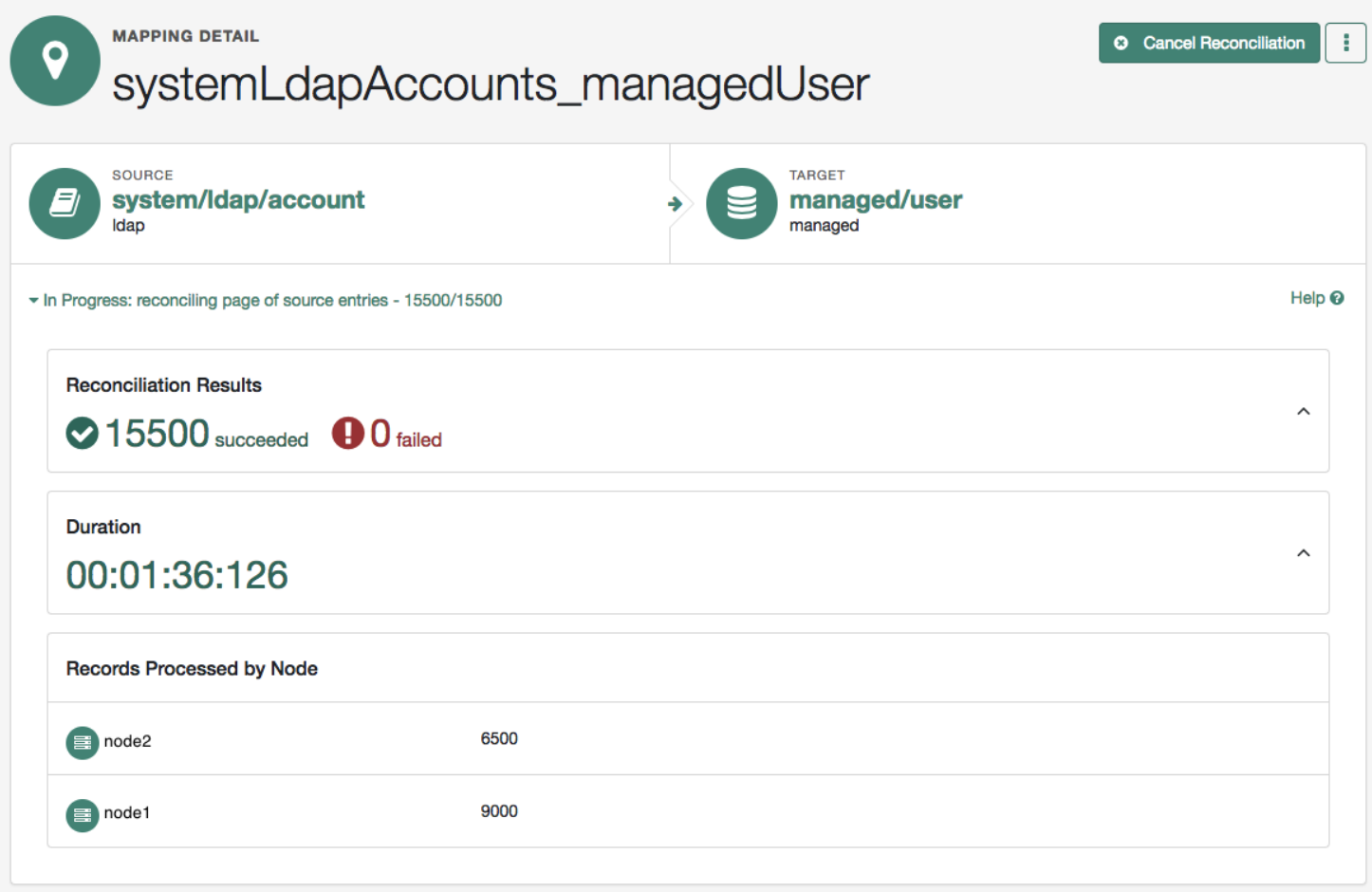


Figure 1. Clustered Reconciliation Results

Cancel a clustered reconciliation

You cancel a clustered reconciliation in the same way as a non-clustered reconciliation, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon/90892122-5ceb-4bbe-86f7-94272df834ad-406025?_action=cancel"
{
  "_id": "90892122-5ceb-4bbe-86f7-94272df834ad-406025",
  "action": "cancel",
  "status": "INITIATED"
}
```

When the cancellation has completed, a query on that reconciliation ID will show the state and stage of the reconciliation as follows:

```
 {
    "_id": "90892122-5ceb-4bbe-86f7-94272df834ad-406025",
    "mapping": "systemLdapAccounts_managedUser",
    "state": "CANCELED",
    "stage": "COMPLETED_CANCELED",
    "stageDescription": "reconciliation aborted.",
    "progress": {
      "source": {
        "existing": {
          "processed": 23500,
          "total": "23500"
        }
      },
      "target": {
        "existing": {
          "processed": 23498,
          "total": "?"
      },
      ...
 }
```

In a clustered environment, *all* reconciliation operations are considered to be "cluster-friendly". This means that even if a mapping is configured as `"clusteredSourceReconEnabled" : false`, you can view the in progress operation on *any* node in the cluster, even if that node is not currently processing the reconciliation. You can also cancel a reconciliation in progress from any node in the cluster.

Tuning reconciliation performance

By default, reconciliation is configured to perform optimally. In some cases, the default optimizations might not be suitable for your deployment. The following sections describe these default optimizations, how they can be configured, and additional methods you can use to improve reconciliation performance.

Correlate empty target sets

To optimize performance, reconciliation does not correlate source objects to target objects if the set of target objects is empty when the correlation is started. This considerably speeds up the process the first time reconciliation is run. You can change this behavior for a specific mapping by adding the `correlateEmptyTargetSet` property to the mapping definition and setting it to `true`. For example:

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

Be aware that this setting will have a performance impact on the reconciliation process.

Prefetch links

All links are queried at the start of reconciliation and the results of that query are used. You can disable the link prefetching so that the reconciliation process looks up each link in the database as it processes each source or target object. To disable link prefetching, add the `prefetchLinks` property to your mapping, and set it to `false`:

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

Be aware that this setting will have a performance impact on the reconciliation process.

Parallel reconciliation threads

By default, reconciliation is multithreaded; numerous threads are dedicated to the same reconciliation run. Multithreading generally improves reconciliation performance. The default number of threads for a single reconciliation run is 10 (plus the main reconciliation thread). Under normal circumstances, you should not need to change this number. However the default might not be appropriate in the following situations:

- The hardware has many cores and supports more concurrent threads. As a rule of thumb for performance tuning, start with setting the thread number to twice the number of cores.

- The source or target is an external system with high latency or slow response times. Threads may then spend considerable time waiting for a response from the external system. Increasing the available threads enables the system to prepare or continue with additional objects.

To change the number of threads, set the `taskThreads` property in your mapping:

```
"mappings" : [
    {
        "name" : "systemCsvfileAccounts_managedUser",
        "source" : "system/csvfile/account",
        "target" : "managed/user",
        "taskThreads" : 20
        ...
    }
]
```

A zero value runs reconciliation as a serialized process, on the main reconciliation thread.

Improve reconciliation query performance

Reconciliation operations are processed in two phases; a *source phase* and a *target phase*. In most reconciliation configurations, source and target queries make a read call to every record on the source and target systems to determine candidates for reconciliation. On slow source or target systems, these frequent calls can incur a substantial performance cost.

To improve query performance in these situations, you can preload the entire result set into memory on the source or target system, or on both systems. Subsequent read queries on known IDs are made against the data in memory, rather than the data on the remote system. For this optimization to be effective, the entire result set must fit into the available memory on the system for which it is enabled.

The optimization works by defining a `sourceQuery` or `targetQuery` in the synchronization mapping that returns not just the ID, but the complete object.

The following example query loads the full result set into memory during the source phase of the reconciliation. The example uses a common filter expression, called with the `_queryFilter` keyword. The query returns the complete object:

```
"mappings" : [
    {
        "name" : "systemLdapAccounts_managedUser",
        "source" : "system/ldap/account",
        "target" : "managed/user",
        "sourceQuery" : {
            "_queryFilter" : "true"
        },
  ...
```

IDM attempts to detect what data has been returned. The autodetection mechanism assumes that a result set that includes three or more fields per object (apart from the `_id` and `rev` fields) contains the complete object.

You can explicitly state whether a query is configured to return complete objects by setting the value of `sourceQueryFullEntry` or `targetQueryFullEntry` in the mapping. The setting of these properties overrides the autodetection mechanism.

Setting these properties to `false` indicates that the returned object is not the complete object. This might be required if a query returns more than three fields of an object, but not the complete object. Without this setting, the autodetect logic would assume that the complete object was being returned. IDM uses only the IDs from this query result. If the complete object is required, the object is queried on demand.

Setting these properties to `true` indicates that the complete object is returned. This setting is typically required only for very small objects, for which the number of returned fields does not reach the threshold required for the auto-detection mechanism to assume that it is a full object. In this case, the query result includes all the details required to pre-load the full object.

The following excerpt indicates that the full objects are returned and that IDM should not autodetect the result set:

```
"mappings" : [
    {
        "name" : "systemLdapAccounts_managedUser",
        "source" : "system/ldap/account",
        "target" : "managed/user",
        "sourceQueryFullEntry" : true,
        "sourceQuery" : {
            "_queryFilter" : "true"
        },
    ...
```

By default, all the attributes defined in the connector configuration are loaded into memory. If your mapping uses only a small subset of the attributes in the connector configuration, you can restrict your query to return only those attributes required for synchronization by using the `_fields` parameter with the query filter.

The following excerpt loads only a subset of attributes into memory, for all users in an LDAP directory.

```
"mappings" : [
    {
        "name" : "systemLdapAccounts_managedUser",
        "source" : "system/ldap/account",
        "target" : "managed/user",
        "sourceQuery" : {
            "_queryFilter" : "true",
            "_fields" : "cn,sn,dn,uid,employeeType,mail"
        },
    ...
```

Note

The default source query for clustered reconciliations and for paged reconciliations is a `queryFilter` that returns the full source objects, not just their IDs. So, source queries for clustered and paged reconciliations are optimized for performance by default.

Paging reconciliation query results

Improve Reconciliation Query Performance describes how to improve reconciliation performance by loading all entries into memory to avoid making individual requests to the external system for every ID. However, this optimization depends on the entire result set fitting into the available memory on the system for which it is enabled. For particularly large data sets (for example, data sets of hundreds of millions of users), having the entire data set in memory might not be feasible.

To alleviate this constraint, you can use reconciliation paging, which breaks down extremely large data sets into chunks. It also lets you specify the number of entries that should be reconciled in each chunk or page.

Reconciliation paging is disabled by default, and can be enabled per mapping. To configure reconciliation paging, set the `reconSourceQueryPaging` property to `true` and set the `reconSourceQueryPageSize` in the synchronization mapping:

```
{
    "mappings" : [
        {
            "name" : "systemLdapAccounts_managedUser",
            "source" : "system/ldap/account",
            "target" : "managed/user",
            "reconSourceQueryPaging" : true,
            "reconSourceQueryPageSize" : 100,
            ...
        }
```

The value of `reconSourceQueryPageSize` must be a positive integer, and specifies the number of entries that will be processed in each page. If reconciliation paging is enabled but no page size is set, a default page size of `1000` is used.

Important

If you are reconciling from a JDBC database using the Database Table connector, you *must* set the `_sortkeys` property in the source query and ensure that the corresponding column is indexed in the database.
The following excerpt of a mapping configures paged reconciliation queries using the Database Table connector:

```
{
    "mappings" : [
        {
            "name" : "systemHrdb_managedUser",
            "source" : "system/db/users",
            "target" : "managed/user",
            "reconSourceQueryPaging" : true,
            "reconSourceQueryPageSize" : 1000,
            "sourceQueryFullEntry" : true,
            "sourceQuery" : {
                "_queryFilter" : "true",
                "_sortKeys" : "email"
            },
        ...
    ]
}
```

Asynchronous reconciliation

Reconciliation can work in tandem with workflows to provide additional business logic to the reconciliation process. You can define scripts to determine the action that should be taken for a particular reconciliation situation. A reconciliation process can launch a workflow after it has assessed a situation, and then perform the reconciliation or some other action.

For example, you might want a reconciliation process to assess new user accounts that need to be created on a target resource. However, new user account creation might require some kind of approval from a manager before the accounts are actually created. The initial reconciliation process can assess the accounts that need to be created, then launch a workflow to request management approval for those accounts. The workflow performs the sync action, based upon the situation assessed during reconciliation (and provided to the workflow through the `ASYNC` action). The workflow then calls the `sync` endpoint with the `performAction` action and triggers a synchronization operation for the specified object.

In this scenario, the defined script returns `ASYNC` for new accounts, and the reconciliation engine does not continue processing the given object. The script then initiates an asynchronous process which, on completion, performs an explicit sync of the source object.

A sample configuration for this scenario is available in `openidm/samples/sync-asynchronous`, and described in Asynchronous reconciliation using workflow.

Configure asynchronous reconciliation Using a workflow

1. Create the workflow definition file (`.xml or .bar` file) and place it in the `openidm/workflow` directory. For more information about creating workflows, refer to Create workflows.
2. Modify the mapping for the situation or situations that should call the workflow. Reference the workflow name in the configuration for that situation.

   For example, the following mapping excerpt calls the `managedUserApproval` workflow if the situation is assessed as `ABSENT`:

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

   In the sample configuration, the workflow makes an explicit call to the `sync` endpoint with the `performAction` action (`openidm.action('sync', 'performAction', content, params)`).

You can also use this kind of explicit synchronization to perform a specific action on a source or target record, regardless of the assessed situation.

To call such an operation over the REST interface, specify the source, and/or target IDs, the mapping, and the action to be taken. The action can be any one of the supported reconciliation actions: `CREATE, UPDATE, DELETE, LINK, UNLINK, EXCEPTION, REPORT, NOREPORT, ASYNC, IGNORE`.

The following example calls the DELETE action on user `bjensen`, whose `_id` in the LDAP directory is `uid=bjensen,ou=People,dc=example,dc=com`. The user is deleted in the target resource; in this case, the repository.

Note

The `_id` must be URL-encoded in the REST call:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/sync?
_action=performAction&sourceId=uid%3Dbjensen%2Cou%3DPeople%2Cdc%3Dexample%2Cdc%3Dcom&mapping=
 systemLdapAccounts_ManagedUser&action=DELETE"
{
    "status": "OK"
}
```

The following example creates a link between a managed object and its corresponding system object. Such a call is useful in the context of manual data association, when correlation logic has linked an incorrect object, or when IDM has been unable to determine the correct target object.

In this example, there are two separate target accounts (`scarter.user` and `scarter.admin`) that should be mapped to the managed object. This call creates a link to the `user` account and specifies a link qualifier that indicates the type of link that will be created:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/sync?_action=performAction&action=LINK
   &sourceId=4b39f74d-92c1-4346-9322-d86cb2d828a8&targetId=scarter.user
   &mapping=managedUser_systemCsvfileAccounts&linkQualifier=user"
{
    "status": "OK"
}
```

For more information about linking to multiple accounts, refer to Map a Single Source Object to Multiple Target Objects.

Import bulk data

The bulk import service lets you import large numbers of external entries over REST. You import entries from a comma-separated values (CSV) file, to a specified managed object type in the IDM repository. Bulk import works as follows:

- Loads bulk CSV entries and stores them temporarily (in the IDM repository) as JSON objects
- Creates a temporary mapping between those entries and the managed object store in the repository
- Performs a reconciliation between the JSON objects and the objects in the repository

Warning

- The bulk import service assumes the CSV file is the *authoritative* data source. If you run an import more than once, the import overwrites all of the properties of the managed object (including timestamps) with the values in the CSV file.
- The bulk import service assumes a singular `type`. If an array of `type` attributes is submitted, the service sets the `type` as the last element of the array.

To import bulk CSV entries into the repository using the REST API, follow these steps:

The first time you upload entries, you must generate a CSV template. The template is essentially an empty CSV file with one header row that matches the managed object type to which you are importing. In most cases, you will be importing data that fits the `managed/user` object model, but you can import any managed object type, such as roles and assignments.

To generate the CSV template, send a GET request to the `openidm/csv/template` endpoint. The following request generates a CSV template for the managed user object type:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/csv/template?resourceCollection=managed/user&_fields=header&_mimeType='text/
plain'"
{
  "_id": "template",
  "header": "\"userName\",\"givenName\",\"sn\",\"mail\",\"description\",\"accountStatus\",
\"telephoneNumber\",
 \"postalAddress\",\"city\",\"postalCode\",\"country\",\"stateProvince\",\"preferences/updates\",
 \"preferences/marketing\""
}
```

The template is generated based on the specified `resourceCollection`, and includes a single header row. The names of each header column are derived from the schema of the managed object type. The template includes only a subset of managed user properties that can be represented by CSV fields.

Only the following managed object properties are included in the header row:

- Properties of type `string`, `boolean`, and `number`
- Properties that do *not* start with an underscore (such as `_id` or `_rev`)

  If you are importing entries to `managed/user`, the bulk import facility assumes that self-service password reset is enabled. This is because the import does not support upload of hashed passwords.

- Properties whose `scope` is not `private`

Set the parameters `_fields=header` and `_mimeType=text/csv` to download the template as a CSV file.

When you have generated the template, export your external data to CSV format, using the headers in the generated template.

You can use the bulk import service with a CSV file up to 50MBytes large and less than 100,000 records. If you need to import a larger file or more records, divide your data into chunks and import each file separately.

You can increase the maximum file size by changing the value of the `maxRequestSizeInMegabytes` property in your `conf/servletfilter-upload.json` file.

You need to use a CSV template to perform a bulk import. For more information, refer to Generate a CSV template.

After formatting your CSV file to match your template's structure, upload the file to the IDM repository with the following request:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--form upload=@/path/to/example-users.csv \
--request POST \
"http://localhost:8080/upload/csv/managed/user?uniqueProperty=userName"
{
  "importUUIDs": [
    "3ebd514f-bdd7-491f-928f-21b72f44e381"
  ]
}
```

`--form` *(-F)*

This option causes `curl` to POST data using the Content-Type `multipart/form-data`, which lets you upload binary files. To indicate that the form content is a file, prefix the file name with an `@` sign.

To import more than one file at once, specify multiple `--form` options, for example:

```
--form upload=@/path/to/example-users-a-j.csv \
--form upload=@/path/to/example-users-k-z.csv \
```

`uniqueProperty` *(required)*

This parameter lets you correlate existing entries, based on a unique value field. This is useful if you need to upload the same file a number of times (for example, if data in the file changes, or if some entries in the file contained errors). You can specify any unique value property here. You can also correlate on more than one property by specifying multiple, comma-delimited unique properties.

A successful upload generates an array of `importUUID`s. You need these UUIDs to perform other operations on the import records.

> **Important**
>
> Note that the endpoint (`upload/csv`) is not an IDM endpoint.

A query on the `csv/metadata` endpoint returns the import ID, the data structure (header fields in the CSV file), a recon ID, and a number of fields indicating the status of the import:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/csv/metadata/?_queryFilter=true"
{
  "result": [
    {
      "_id": "3ebd514f-bdd7-491f-928f-21b72f44e381",
      "_rev": "000000003e8ef4f7",
      "header": [
        "userName",
        "givenName",
        "sn",
        "mail",
        "description",
        "accountStatus",
        "country"
      ],
      "reconId": "2e2cf41a-c4b8-4dda-9d92-6e0af65a15fe-6528",
      "filename": "example-users.csv",
      "resourcePath": "managed/user",
      "total": 1000,
      "success": 1000,
      "failure": 0,
      "created": 1000,
      "updated": 0,
      "unchanged": 0,
      "begin": "2020-04-17T16:31:02.955Z",
      "end": "2020-04-17T16:31:09.861Z",
      "cancelled": false,
      "importDeleted": false,
      "tempRecords": 0,
      "purgedTempRecords": true,
      "purgedErrorRecords": false,
      "authId": "openidm-admin",
      "authzComponent": "internal/user"
    },
    {
      "_rev": "00000000d4392fc8"
    }
  ],
  ...
}
```

Use a query filter to restrict your query to imports to a specific managed object type. The following example queries uploads to the managed user object:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/csv/metadata/?_queryFilter=/resourcePath+eq+"managed/user"'
{
  "result": [
    {
      "_id": "82d9a643-8b03-4cec-86fc-3e09c4c2f01c",
      "_rev": "000000009b3ff60b",
      "header": [
        "userName",
        "givenName",
        "sn",
        "mail",
        "description",
        "accountStatus",
        "country"
      ],
      "reconId": "417dae3b-c939-4191-acbf-6eb1b9e802af-53335",
      "filename": "example-users.csv",
      "resourcePath": "managed/user",
      "total": 1001,
      "success": 1000,
      "failure": 1,
      "created": 0,
      "updated": 0,
      "unchanged": 1000,
      "begin": "2020-04-20T13:12:03.028Z",
      "end": "2020-04-20T13:12:05.222Z",
      "cancelled": false,
      "importDeleted": false,
      "tempRecords": 0,
      "purgedTempRecords": true,
      "purgedErrorRecords": false,
      "authId": "openidm-admin",
      "authzComponent": "internal/user"
    }
  ],
  ...
}
```

The previous example showed the statistics that are returned when you query bulk imports. One of these fields is `"failure": 0,`. If the import was unsuccessful for any records, this `failure` field will have a positive value. You can then download the failed records, examine the failures and correct them in the CSV file, then run the import again.

To download failed records, send a GET request to the endpoint `export/csvImportFailures/importUUID`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--request GET \
--header "Accept-API-Version: resource=1.0" \
"http://localhost:8080/export/csvImportFailures/82d9a643-8b03-4cec-86fc-3e09c4c2f01c"
userName,  givenName,  sn,      mail,     ...,  _importError
emacheke,  Edward,     Macheke, emacheke, ...,  "{code=403, reason=Forbidden, message=Policy validation
   failed, detail={result=false, failedPolicyRequirements=[{policyRequirements=[
   {policyRequirement=VALID_EMAIL_ADDRESS_FORMAT}], property=mail}]}}"
```

The output indicates the failed record or records, and the reason for the failure, in the `_importError` field. In this example, the import failed because of a policy validation error—the email address is not the correct format.

> **Warning**
>
> IDM does not scan for possible CSV injection attacks on uploaded files. *Do not* edit the downloaded CSV file with Microsoft Excel, as this can expose your data to CSV injection.

Cancel an import that is in progress by sending a POST request to the `openidm/csv/metadata/importUUID` endpoint, with the `cancel` action. You might want to cancel an import if the import is taking too long, or if you have noticed problems with the import data, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/csv/metadata/92971c92-67bb-4ae7-b41b-96d249b0b2aa/?_action=cancel"
{
  "status": "OK"
}
```

By default, the timeout for the bulk import servlets is 30 seconds (or `30000` milliseconds). This parameter is set in your `resolver/boot.properties` file as follows:

```
openidm.servlet.timeoutMillis=30000
```

If you are importing a very large number of records, you may need to increase the HTTP request timeout to prevent requests timing out.

In test environments, you can set this parameter to `0` to disable the request timeout. You should *not* disable the timeout in a production environment because no timeout can lead to DDoS attacks where thousands of slow HTTP connections are made.

For a list of all REST endpoints related to bulk import, refer to Bulk import.

Synchronization reference

The synchronization engine is one of the core IDM services. You configure the synchronization service through a `mappings` property that specifies mappings between objects that are managed by the synchronization engine.

```
{
  "mappings": [ object-mapping object, ... ]
}
```

Object-mapping objects

An object-mapping object specifies the configuration for a mapping of source objects to target objects. The `name`, `source`, and `target` properties are mandatory. Other properties are optional or implicit (that is, they have a default value if not set).

```
{
  "correlationQuery"                : script object,
  "correlationScript"               : script object,
  "defaultSourceFields"             : [list of strings],
  "defaultTargetFields"             : [list of strings],
  "displayName"                     : string,
  "enableLinking"                   : boolean,
  "enableSync"                      : boolean,
  "linkQualifiers"                  : [ list of strings ] or script object,
  "links"                           : string,
  "name"                            : string,
  "onCreate"                        : script object,
  "onDelete"                        : script object,
  "onLink"                          : script object,
  "onMapping"                       : script object,
  "onUnlink"                        : script object,
  "onUpdate"                        : script object,
  "optimizeAssignmentSync"          : boolean,
  "policies"                        : [ policy object, ... ],
  "postMapping"                     : script object,
  "properties"                      : [ property object, ... ],
  "queuedSync"                      : { property object },
  "reconAssociation"                : { property object },
  "reconProgressStateUpdateInterval" : integer,
  "reconSourceQueryPageSize"        : integer,
  "reconSourceQueryPaging"          : boolean,
  "reconTargetQueryPageSize"        : integer,
  "reconTargetQueryPaging"          : boolean,
  "result"                          : script object,
  "runTargetPhase"                  : boolean,
  "source"                          : string,
  "sourceCondition"                 : script object or queryFilter string,
  "sourceIdsCaseSensitive"          : boolean,
  "sourceQueryFullEntry"            : boolean,
  "syncAfter"                       : [ list of strings ],
  "target"                          : string,
  "targetIdsCaseSensitive"          : boolean,
  "targetQueryFullEntry"            : boolean,
  "taskThreads"                     : integer,
  "triggerSyncProperties"           : [ list of JSON pointers ],
  "validSource"                     : script object,
  "validTarget"                     : script object
}
```

Mapping object properties

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. For examples of correlation queries, refer to Writing Correlation Queries. The source object is provided in the `source` property in the script scope.

correlationScript

script object, optional

A script that goes beyond a `correlationQuery` of a target system. Used when you need another method to determine which records in the target system relate to the given source record. The syntax depends on the target of the correlation. For information about defining correlation scripts, refer to Writing Correlation Scripts.

defaultSourceFields

list of strings, optional

Allows specifying which fields should be used for read and query requests made on source objects and resource collections.

Set this value to `*` unless you only need a specific set of fields to be returned.

defaultTargetFields

list of strings, optional

Allows specifying which fields should be used for read and query requests made on target objects and resource collections.

Set this value to `*` unless you only need a specific set of fields to be returned.

displayName

string, optional

The mapping name displayed in the UI.

enableLinking

boolean, true or false

Specifies whether links should be maintained between source and target objects for a mapping.

Default : `true`

enableSync

boolean, true or false

Specifies whether automatic synchronization (liveSync and implicit synchronization) should be enabled for a specific mapping. For more information, refer to Disable Automatic Synchronization Operations.

Default : `true`

linkQualifiers

list of strings or script object, optional

Enables mapping of a single source object to multiple target objects.

Example: `"linkQualifiers" : [ "employee", "customer" ]` or

```
"linkQualifiers" : {
        "type" : "text/javascript",
        "globals" : { },
        "source" : "if (returnAll) {
```

```
                    ['contractor', 'employee', 'customer', 'manager']
                  } else {
                    if(object.type === 'employee') {
                      ['employee', 'customer', 'manager']
                    } else {
                      ['contractor', 'customer']
                    }
                  }"
    }
```

If a script object, the script must return a list of strings.

links

string, optional

Enables reuse of the links created in another mapping. Example: `"systemLdapAccounts_managedUser"` reuses the links created by a previous mapping whose `name` is `"systemLdapAccounts_managedUser"`.

name

string, required

Uniquely names the object mapping. Used in the link object identifier.

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, the projected target object in the `target` property, and the link situation that led to the create operation in the `situation` property. Properties on the target object can be modified by the script. If a property value is not set by the script, IDM falls back on the default property mapping configuration. If the script throws an exception, the target object creation is aborted.

onDelete

script object, optional

A script to execute when a target object is to be deleted, after property mappings have been applied. In the root scope, the source object is provided in the `source` property, the target object in the `target` property, and the link situation that led to the delete operation in the `situation` property. If the script throws an exception, the target object deletion 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, and the projected target object in the `target` property.

Note that, although an `onLink` script has access to a copy of the target object, changes made to that copy will not be saved to the target system automatically. If you want to persist changes made to target objects by an `onLink` script, you must explicitly include a call to the action that should be taken on the target object (for example `openidm.create`, `openidm.update` or `openidm.delete`) within the script.

In the following example, when an LDAP target object is linked, the `"description"` attribute of that object is updated with the value `"Active Account"`. A call to `openidm.update` is made within the `onLink` script, to set the value.

```
"onLink" : {
    "type" : "text/javascript",
    "source" : "target.description = 'Active Account';
                openidm.update('system/ldap/account/' + target._id, null, target);"
}
```

If the script throws an exception, target object linking is aborted.

onMapping

script object, optional

A script that is run as part of a mapping. For example, a script that applies effective assignments as part of the mapping. This is run after the mappings have been applied, but before `onUpdate` scripts are executed.

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, and the target object in the `target` property.

Although an `onUnlink` script has access to a copy of the target object, changes made to that copy will not be saved to the target system automatically. If you want to persist changes made to target objects by an `onUnlink` script, you must explicitly include a call to the action that should be taken on the target object (for example, `openidm.create`, `openidm.update` or `openidm.delete`) within the script.

In the following example, when an LDAP target object is unlinked, the `description` attribute of that object is updated with the value `Inactive Account`. A call to `openidm.update` is made within the `onUnlink` script, to set the value.

```
"onUnlink" : {
    "type" : "text/javascript",
    "source" : "target.description = 'Inactive Account';
                openidm.update('system/ldap/account/' + target._id, null, target);"
}
```

If the script throws an exception, target object unlinking 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, the projected target object in the `target` property, and the link situation that led to the update operation in the `situation` property. Any changes that the script makes to the target object will be persisted when the object is finally saved to the target resource. If the script throws an exception, the target object update is aborted.

optimizeAssignmentSync

boolean

When set to `true`, a synchronization event is only triggered by modifications to assignments if those modifications are relevant to that particular mapping. When set to `false` (the default), a synchronization event is triggered for all members of a role or assignment if the relationships or attributes for an assignment are modified.

> **Important**
>
> Don't include `effectiveAssignments` in `triggerSyncProperties` as it negates all performance improvements provided by `optimizeAssignmentSync`.

policies

array of policy objects, optional

Specifies a set of link conditions and associated actions to take in response.

postMapping

script object, optional

A script to execute when sync has been performed on a managed object. This is run after the source object has been successfully synchronized with the target system.

properties

array of property-mapping objects, optional

Specifies mappings between source object properties and target object properties, with optional transformation scripts. refer to Property Object Properties.

queuedSync

list of properties, optional

Specifies the queued synchronization configuration.

reconAssociation

list of properties, optional

Specifies the recon association configuration.

reconProgressStateUpdateInterval

integer, optional

Overrides the number of reconciliation operations required before the reconciliation progress state statistics are persisted to the repository. A value of 50 will write statistics to the repository every 50 operations.

Default : `1024`, minimum : `1`.

reconSourceQueryPageSize

integer

Sets the page size for reconciliation source queries, if paging is enabled.

Default : `10000`

reconSourceQueryPaging

boolean, true or false

Specifies whether paging should be used for reconciliation source queries.

- Default for non-clustered reconciliation : `false`
- Default for clustered reconciliation : `true`

reconTargetQueryPageSize

integer

Sets the page size for reconciliation target queries, if paging is enabled.

Default : `10000`

reconTargetQueryPaging

boolean, true or false

Specifies whether paging should be used for reconciliation target queries.

- Default for non-clustered reconciliation : `false`
- Default for clustered reconciliation : `true`

result

script object, optional

A script executed after a reconciliation finishes.

The variables available to a `result` script are as follows:

`source`	Provides statistics about the source phase of the reconciliation
`target`	Provides statistics about the target phase of the reconciliation
`global`	Provides statistics about the entire reconciliation operation
`context`	Information related to the current operation, such as source and target.
`mappingConfig`	A configuration object representing the mapping being processed.
`reconState`	Provides the state of reconciliation operation; such as, *success*, *failure*, or *active*.

runTargetPhase

boolean, true or false

Specifies whether reconciliation operations should run both the source and target phase. To avoid queries on the target resource, set to `false`.

Default : `true`

source

string, required

Specifies the path of the source object set. Example: `"managed/user"`.

sourceCondition

script object or `queryFilter` string, optional

A script or query filter that determines if a source object should be included in the mapping. If no `sourceCondition` element (or `validSource` script) is specified, all source objects are included in the mapping.

sourceIdsCaseSensitive

boolean, true or false

Consider case sensitivity when linking source IDs. Only effective if this mapping defines links, ignored if the mapping re-uses another mapping's links.

Default : `true`

sourceQueryFullEntry

boolean, true or false, optional

Specifies whether the defined source query returns full object data (`true`) or IDs only (`false`).

No default. If not set in the configuration, IDM will attempt to auto-detect the setting, based on the query results.

syncAfter

list of strings, optional

The specified mapping must be synchronized after all mappings in this list.

target

string, required

Specifies the path of the target object set. Example: `"system/ldap/account"`.

targetIdsCaseSensitive

boolean, true or false

Consider case sensitivity when linking target IDs. Only effective if this mapping defines links, ignored if the mapping re-uses another mapping's links.

Default : `true`

targetQueryFullEntry

Boolean `true` or `false`, optional

Specifies whether the defined target query returns full object data (`true`) or IDs only (`false`).

No default. If not set in the configuration, IDM will attempt to auto-detect the setting, based on the query results.

taskThreads

integer, optional

Sets the number of threads dedicated to the same reconciliation run.

Default : `10`

triggerSyncProperties

list, optional

A list of JsonPointers to fields in the source object whose changes should trigger a synchronization operation.

validSource

script object, optional

A script that determines if a source object is valid to be mapped. The script yields a boolean value: `true` indicates the source object is valid; `false` can be used to defer mapping until some condition is met. In the root scope, the source object is provided in the `source` property. If the script is not specified, then all source objects are considered valid.

validTarget

script object, optional

A script used during the target phase of reconciliation that determines if a target object is valid to be mapped. The script yields a boolean value: `true` indicates that the target object is valid; `false` indicates that the target object should not be included in reconciliation. In the root scope, the target object is provided in the `target` property. If the script is not specified, then all target objects are considered valid for mapping.

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 `""`, 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 condition has an `"object"` property available in root scope, which (if specified) contains the full source object. For example `"source": "(object.email != null)"`. The script is considered to return a boolean value.

default

any value, optional

Specifies the value to assign to the target property if a non-null value is not established by `source` or `transform`. If not specified, the default value is `null`.

Policy objects

A policy object specifies a link condition and the associated actions to take in response.

```
{
  "condition"  : optional, script object,
  "situation"  : string,
  "action"     : string or script object
  "postAction" : optional, script object
}
```

Policy object properties

condition

script object or queryFilter condition, optional

Applies a policy, based on the link type, for example `"condition" : "/linkQualifier eq \"user\""`.

A queryFilter condition can be expressed in two ways—as a string (`"condition" : "/linkQualifier eq \"user\""`) or a map, for example:

```
"condition" : {
    "type" : "queryFilter",
    "filter" : "/linkQualifier eq \"user\""
}
```

It is generally preferable to express a queryFilter condition as a map.

A `condition` script has the following variables available in its scope: `object` and `linkQualifier`.

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.

The `action` script has the following variables available in its scope: `source`, `target`, `sourceAction`, `linkQualifier`, and `recon`.

postAction

script object, optional

Specifies the action to perform after the previously specified action has completed.

The `postAction` script has the following variables available in its scope: `source`, `target`, `action`, `sourceAction`, `linkQualifier`, and `reconID`. `sourceAction` is `true` if the action was performed during the source reconciliation phase, and `false` if the action was performed during the target reconciliation phase. For more information, refer to How Synchronization Situations Are Assessed.

> **Note**
>
> No `postAction` script is triggered if the `action` is either `IGNORE` or `ASYNC`.

Script Object

Script objects take the following form.

```
{
  "type"   : "text/javascript",
  "source": string
}
```

type

string, required

The script type.

IDM supports `"text/javascript"` and `"groovy"`.

source

string, required

Specifies the source code of the script to be executed.

Links

By default, links are maintained between source and target objects in mappings. This behavior is governed by the `enableLinking` property in the mapping. If `enableLinking` is set to `false`, links are not maintained.

You might want to disable linking in the case of a bulk import or during data migration.

To maintain links between source and target objects in mappings, IDM stores an object set in the repository. The object set identifier follows this scheme:

```
links/mapping
```

Here, mapping represents the name of the mapping for which links are managed.

Link entries have the following structure:

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

_id

string

The identifier of the link object.

_rev

string, required

The value of link object's revision.

linkType

string, required

The type of the link. Usually the name of the mapping which created the link.

firstId

string, required

The identifier of the first of the two linked objects.

secondId

string

The identifier of the second of the two linked objects.

Queries

IDM 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.

Reconciliation stages

IDM 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.
    1. Look for a link to a target object in the link object set, and perform a correlation query (if defined).
    2. Determine the link condition, as well as whether a target object can be found.
    3. Determine the action to perform based on the policy defined for the condition.
    4. Perform the action.
    5. Keep track of the target objects for which a condition and action has already been determined.
    6. Write the results.

2. Iterate through all object identifiers for the object set specified as `target`. For each identifier, carry out the following steps:
    1. Find the target in the link object set.

       Determine if the target object was handled in the first phase.
    2. Determine the action to perform based on the policy defined for the condition.
    3. Perform the action.
    4. Write the results.
3. Iterate through all link objects, carrying out the following steps.
    1. If the `reconId` is `"my"`, then skip the object.

       If the `reconId` is not recognized, then the source or the target is missing.
    2. Determine the action to perform based on the policy.
    3. Perform the action.
    4. Store the `reconId` identifer in the mapping to indicate that it was processed in this run.

> **Note**
>
> To optimize a reconciliation operation, the reconciliation process does not attempt to correlate source objects to target objects if the set of target objects is empty when the correlation is started. For information on changing this default behaviour, refer to Correlate Empty Target Sets.

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/csvfile/account/jsmith?_action=liveSync 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]
```

Reconciliation duration metrics

Obtaining the Details of a Reconciliation describes how to obtain the details of a reconciliation run over REST. This section provides more information on the metrics returned when you query the `recon` endpoint. Reconciliation is processed as a series of distinct tasks. The `durationSummary` property indicates the period of time spent on each task. You can use this information to address reconciliation performance bottlenecks.

The following sample output shows the kind of information returned for each reconciliation run:

```
{
    "_id": "3bc72717-a4bb-4871-b936-3a5a560c1a7c-37",
    "duration": 781561,
    "durationSummary": {
        "auditLog": {
            ...
        },
        ...
        "sourceObjectQuery": {
            "count": 100,
            "max": 96,
            "mean": 14,
            "min": 6,
            "stdDev": 16,
            "sum": 1450
        },
        "sourcePagePhase": {
            "count": 1,
            "max": 20944,
            "mean": 20944,
            "min": 20944,
            "stdDev": 0,
            "sum": 20944
        },
        "sourceQuery": {
            "count": 1,
            "max": 120,
            "mean": 120,
            "min": 120,
            "stdDev": 0,
            "sum": 120
        },
        "targetPhase": {
            "count": 1,
            "max": 0,
            "mean": 0,
            "min": 0,
            "stdDev": 0,
            "sum": 0
        },
        "targetQuery": {
            "count": 1,
            "max": 19657,
            "mean": 19657,
            "min": 19657,
            "stdDev": 0,
            "sum": 19657
        }
    },
    ...
}
```


The specific reconciliation tasks that are run depend on the configuration for that mapping. For example, the `sourcePagePhase` is run only if paging is enabled. The `linkQuery` is run only for non-clustered reconciliation operations, because an initial query of all links does not make sense if a single source page query is being run.

Recon tasks

The following list describes all the possible tasks that can be run for a single reconciliation:

`sourcePhase`

This phase runs only for non-clustered, non-paged reconciliations. The total duration (`sum`) is the time spent processing all records on the source system.

`sourcePagePhase`

Queries and processes individual objects in a page, based on their IDs. This phase is run only for clustered reconciliations or for non-clustered reconciliations that have source paging configured. The total duration (`sum`) is the total time spent processing source pages across all cluster nodes. This processing occurs in parallel across all cluster nodes, so it is normal for the `sourcePagePhase` duration to exceed the total reconciliation duration.

`sourceQuery`

Obtains all IDs on the source system, or in a specific source page.

> **Note**
>
> When the `sourceQuery` returns a null paging cookie, indicating that there are no more source IDs to reconcile, the clustered reconciliation process dispatches a scheduled job named `sourcePageCompletionCheck` .
> This job checks for remaining source page jobs on the scheduler. If there are no remaining source page jobs, the `sourcePageCompletionCheck` schedules the target phase. If there are still source page jobs to process, the `sourcePageCompletionCheck` schedules another instance of itself to perform these checks again after a few seconds.
> Because the target phase reconciles all IDs that were not reconciled during the source phase, it cannot start until all of the source pages are complete. Final reconciliation statistics cannot be generated and logged until all source page jobs have completed, so the `sourcePageCompletionCheck` runs even if the target phase is not enabled.

`sourceObjectQuery`

Queries the individual objects on the source system or page, based on their IDs.

`validSourceScript`

Processes any scripts that should be run to determine if a source object is valid to be mapped.

`linkQuery`

Queries any existing links between source and target objects.

This phase includes the following tasks:

`sourceLinkQuery`

Queries any existing links from source objects to target objects.

`targetLinkQuery`

Queries any existing links from target objects that were not processed during the `sourceLinkQuery` phase.

`linkQualifiersScript`

Runs any link qualifier scripts. For more information, refer to Map a Single Source Object to Multiple Target Objects.

`onLinkScript`

Processes any scripts that should be run when source and target objects are linked.

`onUnlinkScript`

Processes any scripts that should be run when source and target objects are unlinked.

`deleteLinkObject`

Deletes any links that are no longer relevant between source and target objects.

`correlationQuery`

Processes any configured correlation queries. For more information, refer to Writing Correlation Queries.

`correlationScript`

Processes any configured correlation scripts. For more information, refer to Writing Correlation Scripts.

`onMappingScript`

For roles, processes the script that applies the effective assignments as part of the mapping.

`activePolicyScript`

Sets the action and active policy based on the current situation.

`activePolicyPostActionScript`

Processes any scripts configured to run after policy validation.

`targetPhase`

The aggregated result for time spent processing records on the target system.

`targetQuery`

Queries all IDs on the target system. The list of IDs is restricted to IDs that have not already been linked to a source ID during the source phase. The target query generates a list of *orphan* IDs that must be reconciled if the target phase is not disabled.

`targetObjectQuery`

Queries the individual objects on the target system, based on their IDs.

`validTargetScript`

Processes any scripts that should be run to determine if a target object is valid to be mapped.

`onCreateScript`

Processes any scripts that should be run when a new target object is created.

`updateTargetObject`

Updates existing linked target objects, based on the configured situations and actions.

`onUpdateScript`

Processes any scripts that should be run when a target object is updated.

`deleteTargetObject`

Deletes any objects on the target resource that must be removed in accordance with the defined synchronization actions.

`onDeleteScript`

Processes any scripts that should be run when a target object is deleted.

`resultScript`

Processes the script that is executed when a reconciliation process has finished.

`propertyMappingScript`

Runs any scripts configured for when source and target properties are mapped.

`postMappingScript`

Processes any scripts that should be run when synchronization has been performed on the managed/user object.

`onReconScript`

Processes any scripts that should be run after source and target systems are reconciled.

`auditLog`

Writes reconciliation results to the audit log.

Metrics Collected

For each phase, the following metrics are collected:

`count`

The number of objects or records processed during that phase. For the `sourcePageQuery` phase, the `count` parameter refers to the page size.

When the `count` statistic of a particular task refers to the number of records being reconciled, the `sum` statistic of that task represents the total time across the total number of threads running in all nodes in the cluster. For example:

```
"updateTargetObject": {
    "count": 1000000,
    "max": 1193,
    "mean": 35,
    "min": 11,
    "stdDev": 0,
    "sum": 35065991
}
```

`max`

The maximum time, in milliseconds, spent processing a record during that phase.

`mean`

The average time, in milliseconds, spent processing a record during that phase.

`min`

The minimum time, in milliseconds, spent processing a record during that phase.

`stdDev`

The standard deviation, which measures the variance of the individual values from the mean.

`sum`

The total amount of time, in milliseconds, spent during that phase.

Security

> *Secure ForgeRock® Identity Management deployments.*

Out-of-the-box, IDM is set up for ease of development and deployment. When you deploy IDM in production, there are specific precautions you should take to minimize security breaches. This guide describes the IDM security mechanisms and strategies you can use to reduce risk and mitigate threats to IDM security.

Certificates and keys

Manage secrets, certificates, and keys.

Passwords

Store and manage passwords securely.

Network

Secure network connections to IDM resources.

Data

Secure IDM stored data.

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

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

Secret stores, certificates, and keys

Encryption makes it possible to protect sensitive data. IDM depends on encryption to negotiate secure network connections, and to keep sensitive data confidential. Encryption in turn depends on keys. IDM stores keys in *secret stores*, and supports the following secret store types:

- File-based keystores
- Property-based secret stores

• Hardware Security Modules (HSM)

> **Caution**
>
> In production environments, avoid using self-signed certificates and certificates associated with insecure ciphers.

Secret stores

Configure secret stores in your project's `conf/secrets.json` file, which has the following default configuration:

```
{
  "stores": [
    {
      "name": "mainKeyStore",
      "class": "org.forgerock.openidm.secrets.config.FileBasedStore",
      "config": {
        "file": "&{openidm.keystore.location|&{idm.install.dir}/security/keystore.jceks}",
        "storetype": "&{openidm.keystore.type|JCEKS}",
        "providerName": "&{openidm.keystore.provider|SunJCE}",
        "storePassword": "&{openidm.keystore.password|changeit}",
        "mappings": [
          {
            "secretId" : "idm.default",
            "types": [ "ENCRYPT", "DECRYPT" ],
            "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}" ]
          },
          ...
        ]
      }
    },
    {
      "name": "mainTrustStore",
      "class": "org.forgerock.openidm.secrets.config.FileBasedStore",
      "config": {
        "file": "&{openidm.truststore.location|&{idm.install.dir}/security/truststore}",
        "storetype": "&{openidm.truststore.type|JKS}",
        "providerName": "&{openidm.truststore.provider|SUN}",
        "storePassword": "&{openidm.truststore.password|changeit}",
        "mappings": [
        ]
      }
    }
  ],
  "populateDefaults": true
}
```

The `mainKeyStore` and `mainTrustStore` properties configure the default secret stores. IDM requires these properties in order to start up. Do not change the property names because they are also provided to third-party products that need a single keystore and a single truststore.

`mainKeyStore`

The main keystore references a Java Cryptography Extension Keystore (JCEKS) located at `/path/to/openidm/security/keystore.jceks`.

`mainTrustStore`

The main truststore references a file-based truststore located at `/path/to/openidm/security/truststore`.

`populateDefaults`

When IDM first starts up, it checks the secrets configuration. If `"populateDefaults": true`, IDM writes a number of encryption keys to the keystore, required to encrypt specific data.

You can manage these keystores and truststores using the `keytool` command, included in your Java installation. For information about the `keytool` command, refer to https://docs.oracle.com/en/java/javase/11/tools/keytool.html.

Each configured store has a `name` and `class`, and the following configuration properties:

`file`

For file-based secret stores, this property references the path to the store file, for example, `&{idm.install.dir}/security/keystore.jceks}`. Hardware security modules do not have a `file` property.

`storetype`

The type of secret store. IDM supports a number of store types, including JCEKS, JKS, PKCS #11, and PKCS #12.

`providerName`

Sets the name of the cryptographic service provider; for example, `SunPKCS11` or `softHSM`. If no provider is specified, the JRE default is used.

`storePassword`

The password to the secret store. For the default IDM keystore and trustore, the password is `changeit`. You should change this password in a production deployment, as described in Changing the Default Keystore Password.

`mappings`

This object lets you map keys and certificates in the secret stores to specific encryption and decryption functionality in IDM. A secrets mapping object has the following structure:

```
{
    "secretId" : "idm.config.encryption",
    "types": [ "ENCRYPT", "DECRYPT" ],
    "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}" ]
}
```

- `secretId` lets you map a secret to one or more aliases and gives an indication of the secret's purpose. For example, `idm.config.encryption` indicates the aliases that are used to encrypt and decrypt sensitive configuration properties.
- `types` indicates what the keys are used for; for example, encryption and decryption of sensitive property values.
- `aliases` are the key aliases in the secret store that are used for this purpose. You can add as many aliases as necessary. The first alias in the list determines which alias is the active one. Active secrets are used for signature generation and encryption.

The aliases in the default keystore are described in Default keystore.

The default secret IDs and the aliases to which they are mapped are listed in Mapping SecretIDs to Key Aliases.

> **Note**
>
> All these properties have a resolvable property value by default; for example `&{openidm.keystore.location}`, that allows you to use *property value substitution*. If no *configuration expression* has been set for a specific property, the value following the vertical bar (`|`) is used. In the following property, the password is `changeit` unless you have set a configuration expression in one of the property resolver locations:
>
> ```
> "storePassword": "&{openidm.keystore.password|changeit}"
> ```

Mapping secretIDs to key aliases

secretId	alias	Description
`idm.default`	`openidm-sym-default`	Encryption keystore for legacy JSON objects that do not contain a `purpose` value in their `$crypto` block
`idm.config.encryption`	`openidm-sym-default`	Encrypts configuration information
`idm.password.encryption`	`openidm-sym-default`	Encrypts managed user passwords
`idm.jwt.session.module.encryption`	`openidm-localhost`	Encrypts JWT session tokens
`idm.jwt.session.module.signing`	`openidm-jwtsessionhmac-key`	Signs JWT session tokens using HMAC
`idm.selfservice.encryption`	`openidm-selfservice-key`	Encrypts JWT self-service tokens
`idm.selfservice.signing`	`selfservice`	Signs JWT session tokens using RSA
`idm.assignment.attribute.encryption`	`openidm-sym-default`	Encrypts confidential assignment attributes

Default keystore

IDM generates a number of encryption keys in a JCEKS keystore the first time the server starts up. These keys map to the secrets defined in Mapping SecretIDs to Key Aliases. Note that the keystore, and the keys, are generated at startup and are not prepackaged. The keys are generated *only* if they do not already exist. You cannot specify custom aliases for these default keys.

To use a different keystore type, such as PKCS #12, create the keystore and generate the keys before you start IDM. This prevents IDM from generating the keys on startup. You can also convert the existing JCEKS keystore to a PKCS #12 keystore. If you use a different keystore type, you must edit the `openidm.keystore.type` property (in the `conf/secrets.json` file) to match the new type.

Use the `keytool` command to list the default encryption keys, as follows:

```
keytool \
-list \
-keystore /path/to/openidm/security/keystore.jceks \
-storepass changeit \
-storetype JCEKS
Keystore type: JCEKS
Keystore provider: SunJCE

Your keystore contains 5 entries

openidm-sym-default, Nov 5, 2019, SecretKeyEntry,
openidm-jwtsessionhmac-key, Nov 5, 2019, SecretKeyEntry,
selfservice, Nov 5, 2019, PrivateKeyEntry,
Certificate fingerprint (SHA-256): E9:0B:BA:FB:58:73:02:FC...:7B
openidm-selfservice-key, Nov 5, 2019, SecretKeyEntry,
openidm-localhost, Nov 5, 2019, PrivateKeyEntry,
Certificate fingerprint (SHA-256): 21:50:6C:90:C7:A7:F7:32...:1B
```

Note

If you are using IDM in a cluster, you must share these keys among all nodes in the cluster. The easiest way to do this is to generate a keystore with the appropriate keys and share the keystore in some way; for example, by using a filesystem that is shared between the nodes.

Change the default keystore password

The default keystore password is `changeit`. You should change this password in a production environment.

Important

Repeat this procedure on each node if you run multiple nodes in a cluster to ensure that the new password is present on all nodes.

1. Shut down the server if it is running.
2. Use the `keytool` command to change the keystore password. The following command changes the keystore password to `newPassword`:

   ```
   keytool \
   -storepasswd \
   -keystore /path/to/openidm/security/keystore.jceks \
   -storetype jceks \
   -storepass changeit
   New keystore password: newPassword
   Re-enter new keystore password: newPassword
   ```

3. Change the passwords of the default encryption keys.

   IDM uses a number of encryption keys, listed in Mapping SecretIDs to Key Aliases, whose passwords are also `changeit` by default. The passwords of each of these keys must match the password of the keystore.

   To get the list of keys in the keystore, run the following command:

```
keytool \
-list \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype jceks \
-storepass newPassword
Keystore type: JCEKS
Keystore provider: SunJCE

Your keystore contains 5 entries

openidm-sym-default, May 4, 2021, SecretKeyEntry,
selfservice, May 4, 2021, PrivateKeyEntry, Certificate fingerprint (SHA-256): fingerprint
openidm-jwtsessionhmac-key, May 4, 2021, SecretKeyEntry,
openidm-localhost, May 4, 2021, PrivateKeyEntry, Certificate fingerprint (SHA-256): fingerprint
openidm-selfservice-key, May 4, 2021, SecretKeyEntry,
```

Change the passwords of each default encryption key as follows:

```
keytool \
-keypasswd \
-alias openidm-localhost \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype jceks \
-storepass newPassword
Enter key password for <openidm-localhost> changeit
New key password for <openidm-localhost>: newPassword
Re-enter new key password for <openidm-localhost>: newPassword
```

```
keytool \
-keypasswd \
-alias openidm-sym-default \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype jceks \
-storepass newPassword
Enter key password for <openidm-sym-default> changeit
New key password for <openidm-sym-default>: newPassword
Re-enter new key password for <openidm-sym-default>: newPassword
```

```
keytool \
-keypasswd \
-alias openidm-selfservice-key \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype jceks \
-storepass newPassword
Enter key password for <openidm-selfservice-key> changeit
New key password for <openidm-selfservice-key>: newPassword
Re-enter new key password for <openidm-selfservice-key>: newPassword
```

```
keytool \
-keypasswd \
-alias selfservice \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype jceks \
-storepass newPassword
Enter key password for <selfservice> changeit
New key password for <selfservice>: newPassword
Re-enter new key password for <selfservice>: newPassword
```

```
keytool \
-keypasswd \
-alias openidm-jwtsessionhmac-key \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype jceks \
-storepass newPassword
Enter key password for <openidm-jwtsessionhmac-key> changeit
New key password for <openidm-jwtsessionhmac-key>: newPassword
Re-enter new key password for <openidm-jwtsessionhmac-key>: newPassword
```

4. Configure a new expression resolver file to store *only* the keystore password:

    1. Create a new directory in `/path/to/openidm/resolver/` that will contain only the properties file for keystore passwords. For example:

    ```
    mkdir /path/to/openidm/resolver/keystore
    ```

    Important

    Substituted properties are not encrypted by default. You *must* therefore secure access to this directory, using the appropriate permissions.

    2. Set the `IDM_ENVCONFIG_DIRS` environment variable to include the new directory:

    ```
    export IDM_ENVCONFIG_DIRS=/path/to/openidm/resolver/,/path/to/openidm/resolver/keystore
    ```

    3. Create a `.json` or `.properties` file in that secure directory, that contains the new keystore password as a resolvable IDM property. For example, add one of the following files to that directory:

    keystorepwd.properties

    ```
    openidm.keystore.password=newPassword
    ```

keystorepwd.json

```
{
    "openidm" : {
        "keystore" : {
            "password" : "newPassword"
        }
    }
}
```

5. Restart IDM.

Encryption key management

Most regulatory requirements mandate that the keys used to decrypt sensitive data be rotated out and replaced with new keys on a regular basis. The main purpose of rotating encryption keys is to reduce the amount of data encrypted with that key, so that the potential impact of a security breach with a specific key is reduced. You can update encryption keys in several ways, including the following:

Manual key rotation

IDM evaluates keys in `secrets.json` sequentially. For example, assume that you have added a new key named `my-new-key` to the keystore. To use this new key to encrypt passwords, you would include `my-new-key` as the *first alias* in the `idm.password.encryption` secret:

```
{
    "secretId" : "idm.password.encryption",
    "types": [ "ENCRYPT", "DECRYPT" ],
    "aliases": [ "my-new-key", "&{openidm.config.crypto.alias|openidm-sym-default}" ]
}
```

The properties that use this key (in this case, passwords) are re-encrypted with the new key the next time the managed object is *updated*. You do not need to restart the server.

Important

If you rotate an encryption key, the *active* encryption key might not be the correct key to use for decryption of properties that have already been encrypted with a previous key.
You must therefore keep all applicable keys in `secrets.json` until every object that is encrypted with old keys have been updated with the latest key.

You can force key rotation on all managed objects by running the `triggerSyncCheck` action on the entire managed object data set. The `triggerSyncCheck` action examines the crypto blob of each object and updates the encrypted property with the correct key.

For example, the following command forces all managed user objects to use the new key:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/managed/user/?_action=triggerSyncCheck"
{
    "status": "OK",
    "countTriggered": 10
}
```

In a large managed object set, the `triggerSyncCheck` action can take a long time to run on only a single node. You should therefore avoid using this action if your data set is large. An alternative to running `triggerSyncCheck` over the entire data set is to iterate over the managed data set and call `triggerSyncCheck` on each individual managed object. You can call this action manually or by using a script.

The following example shows the manual commands that must be run to launch the `triggerSyncCheck` action on all managed users. The first command uses a query filter to return all managed user IDs. The second command iterates over the returned IDs calling `triggerSyncCheck` on each ID:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
"https://localhost:8443/openidm/managed/user?_queryFilter=true&_fields=_id"
{
  "result": [
    {
      "_id": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
      "_rev": "000000004988917b"
    },
    {
      "_id": "55ef0a75-f261-47e9-a72b-f5c61c32d339",
      "_rev": "00000000dd89d671"
    },
    {
      "_id": "998a6181-d694-466a-a373-759a05840555",
      "_rev": "000000006fea54ad"
    },
    ...
  ]
}
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/managed/user/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb?_action=triggerSyncCheck"
```

In large data sets, the most efficient way to achieve key rotation is to use the scheduler service to launch these commands. The following section shows how to use the scheduler service for this purpose.

Scheduled key rotation

This example uses a script to generate multiple scheduled tasks. Each scheduled task iterates over a subset of the managed object set (defined by the `pageSize`). The generated scheduled task then calls another script that launches the `triggerSyncCheck` action on each managed object in that subset.

To schedule key rotation, set up a similar schedule:

1. Create a schedule configuration named `schedule-triggerSyncCheck.json` in your project's `conf` directory. That schedule should look as follows:

```
{
    "enabled" : true,
    "persisted" : true,
    "type" : "cron",
    "schedule" : "0 * * * * ? *",
    "concurrentExecution" : false,
    "invokeService" : "script",
    "invokeContext" : {
        "waitForCompletion" : false,
        "script": {
            "type": "text/javascript",
            "name": "sync/scheduleTriggerSyncCheck.js"
        },
        "input": {
            "pageSize": 2,
            "managedObjectPath" : "managed/user",
            "quartzSchedule" : "0 * * * * ? *"
        }
    }
}
```

   You can change the following parameters of this schedule configuration to suit your deployment:

   `pageSize`

   The number of objects that each generated schedule will handle. This value should be high enough not to create too many schedules. The number of schedules that is generated is equal to the number of objects in the managed object store, divided by the page size.

   For example, if there are 500 managed users and a page size of 100, five schedules will be generated (500/100).

`managedObjectPath`

The managed object set over which the scheduler iterates. For example, `managed/user` if you want to iterate over the managed user object set.

`quartzSchedule`

The schedule at which these tasks should run. For example, to run the task every minute, this value would be `0 * * * * ? *`.

2. The schedule calls a `scheduleTriggerSyncCheck.js` script, located in a directory named `project-dir/script/sync`. Create the `sync` directory, and add the script:

```
var managedObjectPath = object.managedObjectPath;
var pageSize = object.pageSize;
var quartzSchedule = object.quartzSchedule;

var managedObjects = openidm.query(managedObjectPath, {
    "_queryFilter": "true",
    "_fields": "_id"
});

var numberOfManagedObjects = managedObjects.result.length;

for (var i = 0; i < numberOfManagedObjects; i += pageSize) {
    var scheduleId = java.util.UUID.randomUUID().toString();
    var ids = managedObjects.result.slice(i, i + pageSize).map(function(obj) {
        return obj._id
    });
    var schedule = newSchedule(scheduleId, ids);
    openidm.create("/scheduler", scheduleId, schedule);
}

function newSchedule(scheduleId, ids) {
    var schedule = {
        "enabled": true,
        "persisted": true,
        "type": "cron",
        "schedule": quartzSchedule,
        "concurrentExecution": false,
        "invokeService": "script",
        "invokeContext": {
            "waitForCompletion": true,
            "script": {
                "type": "text/javascript",
                "name": "sync/triggerSyncCheck.js"
            },
            "input": {
                "ids": ids,
                "managedObjectPath": managedObjectPath,
                "scheduleId": scheduleId
            }
        }
    };
    return schedule;
}
```

3. Each generated scheduled task calls a script named `triggerSyncCheck.js`. Create the script in your project's `script/sync` directory:

```
var ids = object.ids;
var scheduleId = object.scheduleId;
var managedObjectPath = object.managedObjectPath;

for (var i = 0; i & lt; ids.length; i++) {
    openidm.action(managedObjectPath + "/" + ids[i], "triggerSyncCheck", {}, {});
}

openidm.delete("scheduler/" + scheduleId, null);
```

4. Test the key rotation:

    1. Edit your project's `conf/managed.json` file to return user passwords by default by setting `"scope" : "public"`.

    ```
    "password" : {
        ...
        "encryption" : {
            "purpose" : "idm.password.encryption"
        },
        "scope" : "public",
        ...
    }
    ```

    Because passwords are not returned by default, you will not be able to refer to the new encryption on the password unless you change the property's `scope`.

    2. Perform a GET request to return any managed user entry in your data set:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/managed/user/ccd92204-aee6-4159-879a-46eeb4362807"
{
  "_id" : "ccd92204-aee6-4159-879a-46eeb4362807",
  "_rev" : "0000000009441230",
  "preferences" : {
    "updates" : false,
    "marketing" : false
  },
  "mail" : "bjensen@example.com",
  "sn" : "Jensen",
  "givenName" : "Babs",
  "userName" : "bjensen",
  "password" : {
    "$crypto" : {
      "type" : "x-simple-encryption",
      "value" : {
        "cipher" : "AES/CBC/PKCS5Padding",
        "stableId" : "openidm-sym-default",
        "salt" : "CVrKDuzfzunXfTDbCwU1Rw==",
        "data" : "1I5tWT5aRH/12hf5DgofXA==",
        "keySize" : 16,
        "purpose" : "idm.password.encryption",
        "iv" : "LGE+jnC3ZtyvrE5pfuSvtA==",
        "mac" : "BEXQ1mftxA63dXhJO6dDZQ=="
      }
    }
  },
  "accountStatus" : "active",
  "effectiveRoles" : [ ],
  "effectiveAssignments" : [ ]
}
```

Notice that the user's password is encrypted with the default encryption key (`openidm-sym-default`).

3. Create a new encryption key in the IDM keystore:

```
keytool \
-genseckey \
-alias my-new-key \
-keyalg AES \
-keysize 128 \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype JCEKS
```

4. Shut down the server for keystore to be reloaded.

5. Change your project's `conf/managed.json` file to change the encryption purpose for managed user passwords:

```
"password" : {
    ...
    "encryption" : {
        "purpose" : "idm.password.encryption2"
    },
    "scope" : "public",
    ...
}
```

6. Add the corresponding `purpose` to the `secrets.json` file in the `mainKeyStore` code block:

```
"idm.password.encryption2": {
  "types": [ "ENCRYPT", "DECRYPT" ],
  "aliases": [
    {
      "alias": "my-new-key"
    }
  ]
}
```

7. Restart the server and wait one minute for the scheduled task to start.

8. Perform a GET request again to return the entry of the managed user that you returned previously:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/managed/user/ccd92204-aee6-4159-879a-46eeb4362807"
{
  "_id" : "ccd92204-aee6-4159-879a-46eeb4362807",
  "_rev" : "0000000009441230",
  "preferences" : {
    "updates" : false,
    "marketing" : false
  },
  "mail" : "bjensen@example.com",
  "sn" : "Jensen",
  "givenName" : "Babs",
  "userName" : "bjensen",
  "password" : {
    "$crypto" : {
      "type" : "x-simple-encryption",
      "value" : {
        "cipher" : "AES/CBC/PKCS5Padding",
        "stableId" : "my-new-key",
        "salt" : "CVrKDuzfzunXfTDbCwU1Rw==",
        "data" : "1I5tWT5aRH/12hf5DgofXA==",
        "keySize" : 16,
        "purpose" : "idm.password.encryption2",
        "iv" : "LGE+jnC3ZtyvrE5pfuSvtA==",
        "mac" : "BEXQ1mftxA63dXhJO6dDZQ=="
      }
    }
  },
  "accountStatus" : "active",
  "effectiveRoles" : [ ],
  "effectiveAssignments" : [ ]
}
```

The user password is now encrypted with `my-new-key`.

Change the active alias for managed object encryption

This example describes how to configure and then change the managed object encryption key with a scheduled task. You'll create a new key, set up a managed user, add the key to `secrets.json`, restart IDM, run a `triggerSyncCheck`, and review the result.

1. Create a new key for the IDM keystore in the `security/keystore.jceks` file:

```
keytool \
-genseckey \
-alias my-new-key \
-keyalg AES \
-keysize 128 \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype JCEKS
```

2. For the purpose of this example, in `managed.json`, set `"scope" : "public"` to expose the applied password encryption key.

3. Create a managed user:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "userName": "rsutter",
  "sn": "Sutter",
  "givenName": "Rick",
  "mail": "rick@example.com",
  "telephoneNumber": "6669876987",
  "description": "Another user",
  "country": "USA",
  "password": "Passw0rd"
}' \
"https://localhost:8443/openidm/managed/user/ricksutter"
```

4. Add the newly created `my-new-key` alias to your `conf/secrets.json` file, in the `idm.password.encryption` code block:

```
"idm.password.encryption": {
  "types": [ "ENCRYPT", "DECRYPT" ],
  "aliases": [ "my-new-key", "&{openidm.config.crypto.alias|openidm-sym-default}" ]
}
```

5. To apply the new key to your configuration, shut down and restart IDM.

6. Force IDM to update the key for your users with the `triggerSyncCheck` action:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/managed/user/?_action=triggerSyncCheck"
```

7. Review the result for the newly created user, `ricksutter`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET  \
"https://localhost:8443/openidm/managed/user/ricksutter"
```

8. In the output, you should see the new `my-new-key` encryption key applied to that user's password:

```
...
 "password": {
   "$crypto": {
     "type": "x-simple-encryption",
     "value": {
       "cipher": "AES/CBC/PKCS5Padding",
       "stableId": "my-new-key",
       "salt": "bGyKG3PKmwHONOfxerr1Qg==",
       "data": "6vXZiJ3ZNN/UUnsrT7dTQw==",
       "keySize": 16,
       "purpose": "idm.password.encryption",
       "iv": "doAdtxfWfFbrPIIfubGi5g==",
       "mac": "OML6xd9qvDtD5AvMc1Tc3A=="
     }
   }
 },
...
```

CA-signed certificates

You can use existing CA-signed certificates to secure connections and data by importing the certificates into the keystore, and referencing them your `boot.properties` file. Use the `keytool` command to import an existing certificate into the keystore.

Import CA-signed certificates

The following process imports a CA-signed certificate into the keystore, with the alias example-com. Replace this alias with the alias of your certificate:

1. Stop the server if it is running.

2. Back up your existing `openidm/security/keystore` and `openidm/security/truststore` files.
3. Use the `keytool` command to import your existing certificate into the keystore, substituting your specific information:

`example-cert.p12`	The name of your certificate file.
`srcstorepass`	The certificate password.
`example-com`	The existing certificate alias.
`destination keystore password`	The password for the keystore.

```
keytool \
-importkeystore \
-srckeystore example-cert.p12 \
-srcstoretype PKCS12 \
-srcstorepass changeit \
-srcalias example-com \
-destkeystore keystore.jceks \
-deststoretype JCEKS \
-destalias openidm-localhost
Importing keystore example-cert.p12 to keystore.jceks…
Enter destination keystore password: changeit
```

The keytool command creates a trusted certificate entry with the specified alias and associates it with the imported certificate. The certificate is imported into the keystore with the alias `openidm-localhost`. If you want to use a different alias, you must modify your `resolver/boot.properties` file to reference that alias, as shown in the following step.

> **Note**
>
> The certificate entry password must be the same as the IDM keystore password. If the source certificate entry password is different from the target keystore password, use the `-destkeypass` option with the same value as the `-deststorepass` option to make the certificate password match the target keystore password. If you do not make these passwords the same, no error is generated when you import the certificate (or when you read the certificate entry in the destination keystore), but IDM will fail to start with the following exception:
>
> ```
> java.security.UnrecoverableKeyException: Given final block not properly padded.
> ```

4. If you specified an alias other than `openidm-localhost` for the new certificate, change the value of `openidm.https.keystore.cert.alias` in your `resolver/boot.properties` file to that alias. For example, if your new certificate alias is `example-com`, change the `boot.properties` file as follows:

```
openidm.https.keystore.cert.alias=example-com
```

5. Restart the server.

Delete certificates

When using CA-signed certificates for encryption, it is a best practice to delete *all* unused default certificates from the keystore and truststore using the `keytool` command, as shown in the following examples:

- To delete the `openidm-localhost` certificate from the keystore:

```
keytool \
-delete \
-alias openidm-localhost \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype JCEKS \
-storepass changeit
```

- To delete the `openidm-localhost` certificate from the truststore:

```
keytool \
-delete \
-alias openidm-localhost \
-keystore /path/to/openidm/security/truststore \
-storepass changeit
```

You can use similar commands to delete custom certificates from the keystore and truststore, specifying the certificate alias in the request.

Delete root CA certificates

The Java and IDM truststore files include a number of root CA certificates. Although the probability of a compromised root CA certificate is low, it is a best practice to delete unused root CA certificates.

To review the list of root CA certificates in the IDM truststore:

```
keytool \
-list \
-keystore /path/to/openidm/security/truststore \
-storepass changeit
```

> **Note**
>
> On UNIX/Linux systems, you can find additional lists of root CA certificates in files named `cacerts`. These include root CA certificates associated with your Java environment, typically located in the `${JAVA_HOME}/jre/lib/security/cacerts` directory.

Before making changes to Java environment keystore files, verify any Java-related `cacerts` files are up-to-date and that you have a supported Java version installed.

You can delete root CA certificates with the `keytool` command. For example, to remove the hypothetical `examplecomca2` certificate from the truststore:

```
keytool \
-delete \
-keystore /path/to/openidm/security/truststore \
-storepass changeit \
-alias examplecomca2
```

> **Note**
>
> On Windows systems, you can manage certificates with the Microsoft Management Console (MMC) snap-in tool. For more information, refer to Working With Certificates.

Property-based secret stores

IDM servers can read keys and trusted certificates from properties that contain keys in Privacy-Enhanced Mail (PEM) format.

The following example configures a property-based secret store, and adds an RSA PEM secret whose purpose is to encrypt and decrypt managed user passwords:

1. Add a `PropertyBasedStore` secret store definition to your `conf/secrets.json` file:

```
{
    "name": "pemStore",
    "class": "org.forgerock.openidm.secrets.config.PropertyBasedStore",
    "config": {
        "format": "PEM",
        "algorithm": "RSA",
        "mappings": [
            {
                "secretId": "idm.pem.purpose",
                "types": [
                    "ENCRYPT",
                    "DECRYPT"
                ]
            }
        ]
    }
}
```

2. Create an RSA PEM key:

```
openssl genrsa -out private-key.pem 3072
```

3. Display the private key. For example:

```
more private-key.pem
-----BEGIN RSA PRIVATE KEY-----
MIIG4w…lrDgao
-----END RSA PRIVATE KEY-----
```

4. Use a text editor to convert your certificate to a single line, replacing line breaks with newline characters (`/n`). For example, on UNIX systems:

```
awk 'NF {sub(/\r/, ""); printf "%s\\n",$0;}' private-key.pem
-----BEGIN RSA PRIVATE KEY-----\nMIIG4w…lrDgao\n-----END RSA PRIVATE KEY-----\n%
```

5. Copy the single-line private key and paste it into your `resolver/boot.properties` file, as a value of the `secretId` that you specified in Step 1. For example:

```
idm.pem.purpose=-----BEGIN RSA PRIVATE KEY-----\nMIIG4w...lrDgao\n-----END RSA PRIVATE KEY-----\n%
```

6. Modify the encryption purpose for the managed user `password` in your managed object configuration to use the `PropertyBaseStore` secret store that you created in Step 1:

```
"password" : {
    "title" : "Password",
    "description" : "Password",
    "type" : "string",
    "viewable" : false,
    "searchable" : false,
    "userEditable" : true,
    "encryption" : {
        "purpose" : "idm.pem.purpose",
        "cipher" : "RSA/ECB/OAEPWithSHA-256AndMGF1Padding"
    }
    ...
}
```

IDM now encrypts and decrypts passwords with the RSA PEM key.

Hardware security module (HSM)

This topic demonstrates how to use a PKCS #11 device, such as a hardware security module (HSM), to store the keys used to secure communications. IDM supports retrieval of secrets from HSMs either locally or over the network.

> **Note**
>
> On Windows systems using the 64-bit JDK, the Sun PKCS #11 provider is available *only* from JDK version 1.8b49. If you want to use a PKCS #11 device on Windows, use the 32-bit version of the JDK, or upgrade your 64-bit JDK to version 1.8b49 or higher.

HSM configuration

This section assumes that you have access to an HSM device (or a software emulation of an HSM device, such as SoftHSM) and that the HSM provider has been configured and initialized.

The command-line examples in this section use SoftHSM for testing purposes. Before you start, set the correct environment variable for the SoftHSM configuration, for example:

```
export SOFTHSM2_CONF=/path/to/softhsm/2.0.0/etc/softhsm2.conf
```

Also initialize slot `0` on the provider, with a command similar to the following:

```
softhsm2-util --init-token --slot 0 --label "My token 1"
```

This token initialization requests two PINs—an SO PIN and a user PIN. You can use the SO PIN to reinitialize the token. The user PIN is provided to IDM so that it can interact with the token. Remember the values of these PINs because you will use them later in this section.

The PKCS #11 standard uses a configuration file to interact with the HSM device. The following example shows a basic configuration file for SoftHSM:

```
name = softHSM
library = /path/to/softhsm/2.0.0/lib/softhsm/libsofthsm2.so
slot = 1
attributes(generate, *, *) = {
   CKA_TOKEN = true
}
attributes(generate, CKO_CERTIFICATE, *) = {
   CKA_PRIVATE = false
}
attributes(generate, CKO_PUBLIC_KEY, *) = {
   CKA_PRIVATE = false
}
attributes(*, CKO_SECRET_KEY, *) = {
   CKA_PRIVATE = false
   CKA_EXTRACTABLE = true
}
```

Your HSM configuration file *must* include at least the following settings:

`name`

A suffix to identify the HSM provider. This example uses the `softHSM` provider.

`library`

The path to the PKCS #11 library.

`slot`

The slot number to use, specified as a string. Make sure that the slot you specify here has been initialized on the HSM device.

The `attributes` specify additional PKCS #11 attributes that are set by the HSM. For a complete list of these attributes, refer to the PKCS #11 Reference.

Important

If you are using the JWT Session Module, you *must* set `CKA_EXTRACTABLE = true` for secret keys in your HSM configuration file. For example:

```
attributes(*, CKO_SECRET_KEY, *) = {
   CKA_PRIVATE = false
   CKA_EXTRACTABLE = true
}
```

The HSM provider must allow secret keys to be extractable because the authentication service serializes the JWT Session Module key and passes it to the authentication framework as a base 64-encoded string.

HSM default encryption keys

When IDM first starts up, it generates a number of encryption keys required to encrypt specific data. If you are using an HSM provider, you must generate these keys manually. The secret keys must use an HMAC algorithm.

Note

This procedure assumes that your HSM configuration file is located at `/path/to/hsm/hsm.conf`.

1. The `openidm-sym-default` key is the default symmetric key required to encrypt the configuration. The following command generates that key in the HSM provider. The `-providerArg` must point to the HSM configuration file described in HSM Configuration.

   ```
   keytool \
   -genseckey \
   -alias openidm-sym-default \
   -keyalg HmacSHA256 \
   -keysize 256 \
   -keystore NONE \
   -storetype PKCS11 \
   -providerClass sun.security.pkcs11.SunPKCS11 \
   -providerArg /path/to/hsm/hsm.conf
   Enter keystore password:
   ```

   Enter the password of your HSM device. If you are using SoftHSM, enter your user PIN as the keystore password.

2. The `openidm-selfservice-key` is used by the Self-Service UI to encrypt managed user passwords and other sensitive data. Generate the `openidm-selfservice-key` key:

```
keytool \
-genseckey \
-alias openidm-selfservice-key \
-keyalg HmacSHA256 \
-keysize 256 \
-keystore NONE \
-storetype PKCS11 \
-providerClass sun.security.pkcs11.SunPKCS11 \
-providerArg /path/to/hsm/hsm.conf
Enter keystore password: user PIN
```

Enter the password of your HSM device. If you are using SoftHSM, enter your user PIN as the keystore password.

3. The `openidm-jwtsessionhmac-key` is used by the JWT session module to encrypt JWT session cookies. Generate the JWT session module key:

```
keytool \
-genseckey \
-alias openidm-jwtsessionhmac-key \
-keyalg HmacSHA256 \
-keysize 256 \
-keystore NONE \
-storetype PKCS11 \
-providerClass sun.security.pkcs11.SunPKCS11 \
-providerArg /path/to/hsm/hsm.conf
Enter keystore password: user PIN
```

4. The `openidm-localhost` certificate is used to support SSL/TLS. Generate the certificate:

```
keytool \
-genkey \
-alias openidm-localhost \
-keyalg RSA \
-keysize 2048 \
-keystore NONE \
-storetype PKCS11 \
-providerClass sun.security.pkcs11.SunPKCS11 \
-providerArg /path/to/hsm/hsm.conf
Enter keystore password: user PIN
What is your first and last name?
  [Unknown]:  localhost
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:  OpenIDM Self-Signed Certificate
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=localhost, OU=Unknown, O=OpenIDM Self-Signed Certificate, L=Unknown, ST=Unknown, C=Unknown
correct?
  [no]:  yes
```

5. The `selfservice` certificate secures requests from the End User UI. Generate the certificate:

```
keytool \
-genkey \
-alias selfservice \
-keyalg RSA \
-keysize 2048 \
-keystore NONE \
-storetype PKCS11 \
-providerClass sun.security.pkcs11.SunPKCS11 \
-providerArg /path/to/hsm/hsm.conf
Enter keystore password: user PIN
What is your first and last name?
  [Unknown]:  localhost
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:  OpenIDM Self Service Certificate
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=localhost,O=OpenIDM Self Service Certificate,OU=None,L=None,ST=None,C=None?
  [no]:  yes
```

6. If you are *not* using the HSM provider for the truststore, you must add the certificates generated in the previous two steps to the default IDM truststore.

   If you *are* using the HSM provider for the truststore, you can skip this step.

   To add the `openidm-localhost` certificate to the IDM truststore, export the certificate from the HSM provider, then import it into the truststore:

```
keytool \
-export \
-alias openidm-localhost \
-file exportedCert \
-keystore NONE \
-storetype PKCS11 \
-providerClass sun.security.pkcs11.SunPKCS11 \
-providerArg /path/to/hsm/hsm.conf
Enter keystore password: user PIN
Certificate stored in file exportedCert keytool \
-import \
-alias openidm-localhost \
-file exportedCert \
-keystore /path/to/openidm/security/truststore
Enter keystore password: changeit
Owner: CN=localhost, OU=Unknown, O=OpenIDM Self-Signed Certificate, L=…
Issuer: CN=localhost, OU=Unknown, O=OpenIDM Self-Signed Certificate, L=…
Serial number: 5d2554bd
Valid from: Fri Aug 19 13:11:54 SAST 2016 until: Thu Nov 17 13:11:54 SAST 2016
Certificate fingerprints:
         MD5:  F1:9B:72:7F:7B:79:58:29:75:85:82:EC:79:D8:F9:8D
         SHA1: F0:E6:51:75:AA:CB:14:3D:C5:E2:EB:E5:7C:87:C9:15:43:19:AF:36
         SHA256: 27:A5:B7:0E:94:9A:32:48:0C:22:0F:BB:7E:3C:22:2A:64:B5:45:24:14:70:...
         Signature algorithm name: SHA256withRSA
         Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 7B 5A 26 53 61 44 C2 5A   76 E4 38 A8 52 6F F2 89  .Z&SaD.Zv.8.Ro..
0010: 20 04 52 EE                                        .R.
]
]
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

   The default truststore password is changeit.

Configure IDM to support an HSM provider

To enable IDM to use an HSM provider, make the following configuration changes:

1. In your secret store configuration (`conf/secrets.json`), change the `mainKeyStore` and `mainTrustStore` to reference the HSM. For example:

```
{
  "stores": [
    {
      "name": "mainKeyStore",
      "class": "org.forgerock.openidm.secrets.config.HsmBasedStore",
      "config": {
        "storetype": "&{openidm.keystore.type|PKCS11}",
        "providerName": "&{openidm.keystore.provider|SunPKCS11-softHSM}",
        "storePassword": "&{openidm.keystore.password|changeit}",
        "mappings": [
          {
            "secretId" : "idm.default",
            "types": [ "ENCRYPT", "DECRYPT" ],
            "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}" ]
          },
          {
            "secretId" : "idm.config.encryption",
            "types": [ "ENCRYPT", "DECRYPT" ],
            "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}" ]
          },
          {
            "secretId" : "idm.password.encryption",
            "types": [ "ENCRYPT", "DECRYPT" ],
            "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}" ]
          },
          {
            "secretId" : "idm.jwt.session.module.encryption",
            "types": [ "ENCRYPT", "DECRYPT" ],
            "aliases": [ "&{openidm.https.keystore.cert.alias|openidm-localhost}" ]
          },
          {
            "secretId" : "idm.jwt.session.module.signing",
            "types": [ "SIGN", "VERIFY" ],
            "aliases": [ "&{openidm.config.crypto.jwtsession.hmackey.alias|openidm-jwtsessionhmac-key}" ]
          },
          {
            "secretId" : "idm.selfservice.signing",
            "types": [ "SIGN", "VERIFY" ],
            "aliases": [ "selfservice" ]
          },
          {
            "secretId" : "idm.selfservice.encryption",
            "types": [ "ENCRYPT", "DECRYPT" ],
            "aliases": [ "&{openidm.config.crypto.selfservice.sharedkey.alias|openidm-selfservice-key}" ]
          }
        ]
      }
    },
    {
      "name": "mainTrustStore",
      "class": "org.forgerock.openidm.secrets.config.HsmBasedStore",
      "config": {
        "storetype": "&{openidm.keystore.type|PKCS11}",
        "providerName": "&{openidm.keystore.provider|SunPKCS11-softHSM}",
        "storePassword": "&{openidm.keystore.password|changeit}",
        "mappings": [
        ]
```

```
        }
      }
    ],
    "populateDefaults": false
}
```

Important

The `"populateDefaults": false` turns off the default key generation. This setting is *required* for an HSM key provider.

2. In the IDM Java security file (`conf/java.security`), Specify the location of your PKCS #11 configuration file. For example:

```
security.provider.14=SunPKCS11 /path/to/pkc11/config/pkcs11.conf
```

Templates for the `pkcs11.conf` file are included in your PKCS package.

You should now be able to start IDM with the keys in the HSM provider.

FIPS 140-2 compliance

Important

In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
Learn more in What's new.

To achieve FIPS 140-2 compliance, configure the Bouncy Castle FIPS libraries with IDM. This enables the use of the Bouncy Castle FIPS keystore and security provider in FIPS-approved mode.

Bouncy Castle FIPS is useful when dealing with government data, where meeting the FIPS 140-2 security requirement is necessary for regulatory compliance.

Note

Bouncy Castle FIPS is less performant than other keystores. The destroyable keys cannot be cached and must be read from the keystore with every use.

To configure IDM to use Bouncy Castle FIPS:

1. Download the Bouncy Castle libraries.
2. Enable the Bouncy Castle FIPS provider in the JVM.
3. Create the IDM cryptographic keys.
4. Provide the JVM to IDM.

5. Configure the Bouncy Castle keystore in `secrets.json`.

Download the Bouncy Castle libraries

> **Important**
>
> The IDM CLI does not work when using Bouncy Castle FIPS.

To use Bouncy Castle FIPS with IDM, download the libraries:

1. Download the following libraries from Bouncy Castle to the server/machine where IDM is deployed:

File	Description
`bc-fips-latestVersionNumber.jar` (1)	Contains the Bouncy Castle FIPS security provider implementation.
`bcpkix-fips-latestVersionNumber.jar` (2)	Provides FIPS support for cert generation.
`bctls-fips-latestVersionNumber.jar` (3)	Provides TLS support using FIPS compliance.

(1) The tested version is `bc-fips-1.0.2.3.jar`.

(2) The tested version is `bcpkix-fips-1.0.7.jar`.

(3) The tested version is `bctls-fips-1.0.14.jar`.

2. Copy the downloaded files to `/path/to/openidm/bundle`.
3. Restart IDM.

Enable the Bouncy Castle FIPS provider in the JVM

To enable the Bouncy Castle FIPS provider in your JVM, do one of the following:

- Add Bouncy Castle providers to the existing JVM
- Add Bouncy Castle providers to IDM `conf/java.security`
- Build a distribution of the JVM that supports Bouncy Castle

Add Bouncy Castle providers to the existing JVM

If the existing JVM supports Bouncy Castle, then you can add the security providers to the JVM.

Add the Bouncy Castle security providers to `$JAVA_HOME/conf/security/java.security`:

```
security.provider.13=org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider C:HYBRID;ENABLE{All};
security.provider.14=org.bouncycastle.jsse.provider.BouncyCastleJsseProvider
```

> **Note**
>
> The `security.provider.13` and `security.provider.14` keys are JVM specific and must be the next 2 values in the security providers list.

Add Bouncy Castle providers to IDM `conf/java.security`

If the existing JVM supports Bouncy Castle, then you can add the security providers to the `/path/to/openidm/conf/java.security`. This file provides additions to the `$JAVA_HOME/conf/security/java.security` file.

Add the Bouncy Castle security providers to `/path/to/openidm/conf/java.security`:

```
security.provider.13=org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider C:HYBRID;ENABLE{All};
security.provider.14=org.bouncycastle.jsse.provider.BouncyCastleJsseProvider
```

> **Note**
>
> The `security.provider.13` and `security.provider.14` keys are JVM specific and must be the next 2 values in the security providers list.

Build a distribution of the JVM that supports Bouncy Castle

If the existing JVM doesn't support Bouncy Castle, you must create a new JVM distribution.

1. Build a new distribution:

   ```
   $JAVA_HOME/bin/jlink \
   --no-header-files \
   --no-man-pages \
   --compress=2 \
   --module-path /Users/bjensen/Downloads \
   --add-modules
   java.base,java.compiler,java.datatransfer,java.desktop,java.instrument,java.logging,java.management,java.manag
   ement.rmi,java.naming,java.net.http,java.prefs,java.rmi,java.scripting,java.se,java.security.jgss,java.securit
   y.sasl,java.smartcardio,java.sql,java.sql.rowset,java.transaction.xa,java.xml,java.xml.crypto,jdk.accessibilit
   y,jdk.aot,jdk.attach,jdk.charsets,jdk.compiler,jdk.crypto.cryptoki,jdk.crypto.ec,jdk.dynalink,jdk.editpad,jdk.
   hotspot.agent,jdk.httpserver,jdk.internal.ed,jdk.internal.jvmstat,jdk.internal.le,jdk.internal.opt,jdk.interna
   l.vm.ci,jdk.internal.vm.compiler,jdk.internal.vm.compiler.management,jdk.jartool,jdk.javadoc,jdk.jcmd,jdk.jcon
   sole,jdk.jdeps,jdk.jdi,jdk.jdwp.agent,jdk.jfr,jdk.jlink,jdk.jshell,jdk.jsobject,jdk.jstatd,jdk.localedata,jdk.
   management,jdk.management.agent,jdk.management.jfr,jdk.naming.dns,jdk.naming.ldap,jdk.naming.rmi,jdk.net,jdk.p
   ack,jdk.rmic,jdk.scripting.nashorn,jdk.scripting.nashorn.shell,jdk.sctp,jdk.security.auth,jdk.security.jgss,jd
   k.unsupported,jdk.unsupported.desktop,jdk.xml.dom,jdk.zipfs,org.bouncycastle.fips.core,org.bouncycastle.fips.t
   ls \
   --output /location/to/bouncy/castle/jvm --ignore-signing-information
   ```

   A customized JVM is created at the output location /location/to/bouncy/castle/jvm you specified.

2. Add the Bouncy Castle security providers to `/location/to/bouncy/castle/jvm/conf/security/java.security`:

```
security.provider.13=org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider C:HYBRID;ENABLE{All};
security.provider.14=org.bouncycastle.jsse.provider.BouncyCastleJsseProvider
```

> **Note**
>
> The `security.provider.13` and `security.provider.14` keys are JVM specific and must be the next 2 values in the security providers list.

Create the IDM Bouncy Castle keystore and cryptographic keys

> **Note**
>
> Before you create the cryptographic keys, you must Enable the Bouncy Castle FIPS provider in the JVM.

To create the necessary IDM cryptographic keys:

1. Create the Bouncy Castle keystore. This can be done in conjunction with creating the first cryptographic key:

```
keytool \
-genseckey \
-alias openidm-sym-default \
-keyalg aes \
-keysize 256 \
-keystore /location/to/keystore.bcfks \
-storepass changeit -storetype BCFKS \
-provider org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
```

This creates the `openidm-sym-default` key in a keystore called /location/to/keystore.bcfks while also creating that keystore if it does not exist.

> **Note**
>
> You must use the JVM specific keytool that the Bouncy Castle security provider uses.
> For example, if you enable the security providers in the system default JVM, you must use the system default keytool command. If you create a custom JVM, you must use the keytool command for where that JVM is located.
> The keytool command is in the `bin` directory of the JVM Java home.
> Failure to use the `keytool` command you configure for Bouncy Castle results in the following error:
>
> ```
> keytool error: java.lang.Exception: Provider
> "org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider" not found
> ```

2. Create the remaining keys:

Create the `openidm-selfservice-key`

```
keytool \
-genseckey \
-alias openidm-selfservice-key \
-keyalg aes \
-keysize 256 \
-keystore /location/to/keystore.bcfks \
-storepass changeit \
-storetype BCFKS \
-provider org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
```

Create the `openidm-jwtsessionhmac-key`

```
keytool \
-genseckey \
-alias openidm-jwtsessionhmac-key \
-keyalg aes \
-keysize 256 \
-keystore /location/to/keystore.bcfks \
-storepass changeit \
-storetype BCFKS \
-provider org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
```

Create the `openidm-localhost` key

```
keytool \
-genkey \
-alias openidm-localhost \
-keyalg RSA \
-keysize 2048 \
-keystore /location/to/keystore.bcfks \
-storepass changeit \
-storetype BCFKS \
-provider org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
```

Create the `selfservice` key

```
keytool \
-genkey \
-alias selfservice \
-keyalg RSA \
-keysize 2048 \
-keystore /location/to/keystore.bcfks \
-storepass changeit \
-storetype BCFKS \
-provider org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
```

Provide the JVM to IDM

If you create a custom JVM location, you must to provide that JVM to IDM in the `/path/to/openidm/startup.sh` file.

By default, IDM uses the system Java and falls back to using the `JAVA_HOME` if the system Java is not defined:

Default `startup.sh` file

```
if which java &>/dev/null; then
    JAVA=java
elif [ -n "$JAVA_HOME" ] && [ -x "$JAVA_HOME/bin/java" ];  then
    JAVA="$JAVA_HOME/bin/java"
else
    echo JAVA_HOME not available, Java is needed to run IDM
    echo Please install Java and set JAVA_HOME accordingly
    exit 1
fi
```

To configure IDM to use the `JAVA_HOME` you set, change the `startup.sh` file to the following:

Modified `startup.sh` file

```
if [ -n "$JAVA_HOME" ] && [ -x "$JAVA_HOME/bin/java" ];  then
    JAVA="$JAVA_HOME/bin/java"
else
    echo JAVA_HOME not available, Java is needed to run IDM
    echo Please install Java and set JAVA_HOME accordingly
    exit 1
fi
```

Configure the Bouncy Castle keystore in `secrets.json`

After you add the Bouncy Castle security providers and create the keystore and keys, you must replace the default IDM keystore with the new Bouncy Castle keystore in `/path/to/openidm/conf/secrets.json`:

```
{
    "name" : "mainKeyStore",
    "class" : "org.forgerock.openidm.secrets.config.FileBasedStore",
    "config" : {
        "file" : "&{idm.install.dir}/security/keystore.bcfks",
        "storetype" : "BCFKS",
        "providerName" : "BCFIPS",
        "storePassword" : "changeit",
        "mappings" : [
            {
                "secretId" : "idm.default",
                "types" : [
                    "ENCRYPT",
                    "DECRYPT"
                ],
                "aliases" : [
                    "&{openidm.config.crypto.alias|openidm-sym-default}"
                ]
            },
            {
                "secretId" : "idm.config.encryption",
                "types" : [
                    "ENCRYPT",
                    "DECRYPT"
                ],
                "aliases" : [
                    "&{openidm.config.crypto.alias|openidm-sym-default}"
                ]
            },
            {
                "secretId" : "idm.password.encryption",
                "types" : [
                    "ENCRYPT",
                    "DECRYPT"
                ],
                "aliases" : [
                    "&{openidm.config.crypto.alias|openidm-sym-default}"
                ]
            },
            {
                "secretId" : "idm.assignment.attribute.encryption",
                "types" : [
                    "ENCRYPT",
                    "DECRYPT"
                ],
                "aliases" : [
                    "&{openidm.config.crypto.alias|openidm-sym-default}"
                ]
            }
        ]
    }
},
```

IDM is now configured to start using the Bouncy Castle keystore.

Disable Bouncy Castle FIPS-approved mode

By default, IDM turns on Bouncy Castle in FIPS-approved mode. This makes Bouncy Castle FIPS 140-2 compliant.

IDM sets the configuration in `/path/to/openidm/startup.sh` and `/path/to/openidm/bin/docker-entrypoint.sh` using the following property:

```
org.bouncycastle.fips.approved_only=true
```

To disable FIPS-approved mode, change `org.bouncycastle.fips.approved_only` to `false`.

> **Note**
>
> In `startup.sh` and `docker-entrypoint.sh` there is also the property `org.bouncycastle.jca.enable_jks=true`. This property enables the Java keystore (JKS format) for FIPS. In order to maintain compliance, the keystore can only be used for reading a Java keystore that contains certificates. IDM sets this property to `true` by default.

> **Note**
>
> These settings must take place early in the IDM start process per Bouncy Castle's documentation.

Passwords

IDM provides password management features that help you enforce password policies, limit the number of passwords users must remember, and allow users to reset and change their passwords.

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.

IDM enforces password policy rules as part of the general policy service. The default password policy applies the following rules to passwords as they are created and updated:

- A password property is required for any user object.
- The value of a password cannot be empty.
- The password must include at least one capital letter.
- The password must include at least one number.
- The minimum length of a password is 8 characters.
- The password cannot contain the user name, given name, or family name.

You can change these validation requirements, or include additional requirements, by configuring the policy for passwords.

Passwords are validated in several situations:

Password change and password reset

Password *change* refers to users changing their own passwords. Password *reset* refers to an administrator setting a user or account password on behalf of a user.

By default, IDM validates password values as they are provisioned.

Password recovery

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

Password expiration

You can use workflows to ensure that users are able to change expiring passwords or to reset expired passwords.

Multiple passwords per linked resource

You can store multiple passwords in a single managed user entry to enable synchronization of different passwords on different external resources.

To store multiple passwords, extend the managed user schema to include additional properties for each target resource. You can set separate policies on each of these new properties, to ensure that the stored passwords adhere to the password policies of the specific external resources.

Random passwords

In certain situations, you might want to generate a random password when users are created.

You can customize your user creation logic to include a randomly generated password that complies with the default password policy. This functionality is included in the default crypto script, `bin/defaults/script/crypto.js`, but is not invoked by default. For an example of how this functionality might be used, refer to the `openidm/bin/defaults/script/onCreateUser.js` script. The following section of that file (commented out by default) means that users created through the admin UI, or directly over the REST interface, will have a randomly generated password added to their entry:

```
if (!object.password) {

    // generate random password that aligns with policy requirements
    object.password = require("crypto").generateRandomString([
        { "rule": "UPPERCASE", "minimum": 1 },
        { "rule": "LOWERCASE", "minimum": 1 },
        { "rule": "INTEGERS", "minimum": 1 },
        { "rule": "SPECIAL", "minimum": 1 }
    ], 16);

}
```

> **Note**
>
> The changes made to scripts take effect after the time set in the `recompile.minimumInterval`, described in Script configuration.

The generated password can be encrypted or hashed, in accordance with the managed user schema, defined in `conf/managed.json`. For more information, refer to Encoding Attribute Values. Synchronizing hashed passwords is not supported.

You can use this random string generation in a number of situations. Any script handler that is implemented in JavaScript can call the `generateRandomString` function.

`password` property

To use a property other than the default `password` property to store passwords, you must change the following files:

`policy.json`

If you want to enforce password validation rules on a different property, change the `password` property in this file.

`managed.json`

Modify the `password` object in this file, which also includes password complexity policies.

`sync.json`

If you change the `password` property, make sure that you limit the change to the appropriate system, designated as `source` or `target`.

Every UI file that includes `password` *as a property name*

Whenever there's a way for a user to enter a password, the associated `HTML` page will include a password entry. For example, the `LoginTemplate.html` file includes the `password` property. A full list of default files with the `password` property include:

- `_passwordFields.html`
- `_resetPassword.html`
- `ConfirmPasswordDialogTemplate.html`
- `EditPasswordPageView.html`
- `LoginTemplate.html`
- `MandatoryPasswordChangeDialogTemplate.html`
- `resetStage-initial.html`
- `UserPasswordTab.html`

> **Note**
>
> This list does not include any created custom UI files.

Email rate limiting

No rate limiting is applied to password reset emails, or any emails sent by the IDM server. This means that an attacker can potentially spam a known user account with an infinite number of emails, filling that user's inbox. In the case of password reset, the spam attack can obscure an actual password reset attempt.

In a production environment, you must configure email rate limiting through the network infrastructure in which IDM runs. Configure the network infrastructure to detect and prevent frequent repeated requests to publicly accessible web pages, such as the password reset page. You can also handle rate limiting within your email server.

Network connections

This topic explains how to secure incoming connections and ports. As a general precaution in production environments, avoid communication over insecure HTTP.

Use TLS/SSL

Use TLS/SSL to access IDM, ideally with mutual authentication so that only trusted systems can invoke each other. TLS/SSL protects data on the network. Mutual authentication with strong certificates, imported into the truststore and keystore of each application, provides a level of confidence for trusting application access.

Restrict REST access to the HTTPS port

In a production environment, you should restrict REST access to a secure port:

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

1. Edit your project's `conf/jetty.xml` file:

   Comment out or delete the `<Call name="addConnector">` code block that includes the `openidm.port.http` property.

   > **Note**
   >
   > Do not delete the `<Call name="addConnector">` code blocks that contain the `openidm.port.https` and `openidm.port.mutualauth` properties.

2. Edit `resolver/boot.properties` :
   - Set the `openidm.port.https` port number.
   - Set the `openidm.port.mutualauth` port number.
   - Add the property `openidm.https.enabled=true` .

Use a certificate to secure REST access over HTTPS. You can use self-signed certificates in a test environment. In production, all certificates should be signed by a certificate authority. The examples in this guide assume a CA-signed certificate named `ca-cert.pem` .

Protect sensitive REST interface URLs

Anything attached to the router is accessible with the default policy, including the repository. If you do not need such access, deny it in the authorization policy to reduce the attack surface.

In addition, you can deny direct HTTP access to system objects in production, particularly access to `action`. As a rule of thumb, do not expose anything that is not used in production.

For an example that shows how to protect sensitive URLs, refer to Configure Access Control in access.json.

Enable HTTP Strict-Transport-Security

HTTP Strict-Transport-Security (HSTS) is a web security policy that forces browsers to make secure HTTPS connections to specified web applications. HSTS can protect websites against passive eavesdropper and active man-in-the-middle attacks.

IDM provides an HSTS configuration, but it is disabled by default. To enable HSTS, locate the following excerpt in your `conf/jetty.xml` file:

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

```
<New id="tlsHttpConfig" class="org.eclipse.jetty.server.HttpConfiguration">
    ...
    <Call name="addCustomizer">
        <Arg>
            <New class="org.eclipse.jetty.server.SecureRequestCustomizer">
                <!-- Enable SNI Host Check when true -->
                <Arg name="sniHostCheck" type="boolean">true</Arg>
                <!-- Enable Strict-Transport-Security header and define max-age when >= 0 seconds -->
                <Arg name="stsMaxAgeSeconds" type="long">-1</Arg>
                <!-- If enabled, add includeSubDomains to Strict-Transport-Security header when true -->
                <Arg name="stsIncludeSubdomains" type="boolean">false</Arg>
            </New>
        </Arg>
    </Call>
...
```

Set the following arguments:

`stsMaxAgeSeconds`

This parameter sets the length of time, in seconds, that the browser should remember that a site can only be accessed using HTTPS.

For example, the following setting applies the HSTS policy and remains in effect for an hour:

```
<Arg name="stsMaxAgeSeconds" type="long">3600</Arg>
```

`stsIncludeSubdomains`

If this parameter is `true`, the HSTS policy is applied to the domain of the issuing host as well as its subdomains:

```
<Arg name="stsIncludeSubdomains" type="boolean">true</Arg>
```

For more information about HSTS, click here.

Restrict the HTTP payload size

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

Restricting the size of HTTP payloads can protect the server against large payload HTTP DDoS attacks. IDM includes a servlet filter that limits the size of an incoming HTTP request payload, and returns a `413 Request Entity Too Large` response when the maximum payload size is exceeded.

By default, the maximum payload size is 5MB. You can configure the maximum size in your project's `conf/servletfilter-payload.json` file. That file has the following structure by default:

```
{
    "classPathURLs" : [ ],
    "systemProperties" : { },
    "requestAttributes" : { },
    "scriptExtensions" : { },
    "initParams" : {
        "maxRequestSizeInMegabytes" : 5
    },
    "urlPatterns" : [
        "/*"
    ],
    "filterClass" : "org.forgerock.openidm.jetty.LargePayloadServletFilter"
}
```

Change the value of the `maxRequestSizeInMegabytes` property to set a different maximum HTTP payload size.

Deploy securely behind a load balancer

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

IDM prevents URL-hijacking, with the following code block in the `conf/jetty.xml` file:

```
<Call name="addCustomizer">
  <Arg>
    <New class="org.eclipse.jetty.server.SecureRequestCustomizer">
      <!-- Enable SNI Host Check when true -->
      <Arg name="sniHostCheck" type="boolean">true</Arg>
      <!-- Enable Strict-Transport-Security header and define max-age when >= 0 seconds -->
      <Arg name="stsMaxAgeSeconds" type="long">-1</Arg>
      <!-- If enabled, add includeSubDomains to Strict-Transport-Security header when true -->
      <Arg name="stsIncludeSubdomains" type="boolean">false</Arg>
    </New>
  </Arg>
</Call>
```

If you are deploying IDM behind a system such as a load balancer, firewall, or a reverse proxy, you must uncomment the next section in `jetty.xml` , so that Jetty honors `X-Forwarded-Host` headers:

```
 <Call name="addCustomizer">
    <Arg>
        <New class="org.eclipse.jetty.server.ForwardedRequestCustomizer">
            <Set name="forcedHost">
                <Call class="org.forgerock.openidm.jetty.Param" name="getProperty">
                    <Arg>openidm.host</Arg>
                </Call>:<Call class="org.forgerock.openidm.jetty.Param" name="getProperty">
                    <Arg>openidm.port.https</Arg>
                </Call>
            </Set>
        </New>
    </Arg>
</Call>
```

Connect to IDM through a proxy server

1. Add the following JVM parameters to the value of `OPENIDM_OPTS` in your startup script (`startup.sh` or `startup.bat`):

   -Dhttps.proxyHost

   Hostname or IP address of the proxy server; for example, `proxy.example.com` or `192.168.0.1` .

-Dhttps.proxyPort

Port number used by IDM; for example, `8443` or `9443`.

```
# Only set OPENIDM_OPTS if not already set
[ -z "$OPENIDM_OPTS" ] && OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dhttps.proxyHost=localhost -
Dhttps.proxyPort=8443"
```

2. Enable the `ForwardedRequestCustomizer` class so that Jetty honors `X-Forwarded-` headers.

   To enable the class, uncomment the following excerpt in your `conf/jetty.xml` file:

> **Important**
>
> In IDM 7.3.2 and later, the embedded Jetty web server supports Jetty 12. Future releases of IDM 7.3.x are only compatible with Java 17. Jetty 12 requires this change.
> When serving SSL requests, Jetty 12 checks that the incoming host header matches the server certificate's subject and returns a `400 Bad Request` error on a mismatch. If you're upgrading to IDM 7.3.2 or later, you must ensure your IDM server certificate subject matches the host name used by your deployment.
> Learn more in What's new.

```
<Call name="addCustomizer">
    <Arg>
        <New class="org.eclipse.jetty.server.ForwardedRequestCustomizer">
            <Set name="forcedHost">
                <Call class="org.forgerock.openidm.jetty.Param" name="getProperty">
                    <Arg>openidm.host</Arg>
                </Call>:
                <Call class="org.forgerock.openidm.jetty.Param" name="getProperty">
                    <Arg>openidm.port.https</Arg>
                </Call>
            </Set>
        </New>
    </Arg>
</Call>
```

   Learn more about this class in Jetty documentation.

Protect against Cross-Site Request Forgery

IDM provides a filter to protect against Cross-Site Request Forgery (CSRF). The filter is disabled by default. To enable it, set the following property in `resolver/boot.properties`:

```
openidm.csrfFilter.enabled=true
```

The filter requires the client to send a CSRF cookie (`X-CSRF-Token`) on every request. By default, this cookie is the JWT session cookie (`session-jwt`), obtained on authentication. If your client uses a different cookie for authentication, set the name of that cookie in the following property in `resolver/boot.properties`:

```
openidm.csrfFilter.authCookieName=session-jwt
```

If there are HTTP request paths that the CSRF filter should always allow, set these paths as comma-separated values of the `openidm.csrfFilter.pathWhitelistCSV` property in `resolver/boot.properties`. For example:

```
openidm.csrfFilter.pathWhitelistCSV=/openidm/example,/openidm/my-project
```

IDM data

Beyond relying on end-to-end availability of TLS/SSL to protect data, IDM also supports explicit encryption of data that goes on the network. This can be important if the TLS/SSL termination happens prior to the final endpoint.

IDM also supports encryption of data stored in the repository, using the symmetric keys specified in `conf/secrets.json`. This protects against some attacks on the data store. Explicit table mapping is supported for encrypted string values.

IDM automatically encrypts sensitive data (such as passwords) in configuration files, and replaces cleartext values when the system first reads the configuration file. Take care with configuration files that contain clear text values that IDM has not yet read and encrypted.

Encode attribute values

There are two ways to encode attribute values for managed objects—reversible encryption and salted hashing algorithms. Attribute values that might be encoded include passwords, authentication questions, credit card numbers, and social security numbers. If passwords are already encoded on the external resource, they are generally excluded from the synchronization process. For more information, refer to Passwords.

You configure attribute value encoding, per schema property, in the managed object configuration. The following sections show how to use reversible encryption and salted hash algorithms to encode attribute values.

Reversible encryption

The following managed object configuration encrypts and decrypts the `password` attribute using the default symmetric key:

```
{
    "objects" : [
        {
            "name" : "user",
            ...
            "schema" : {
                ...
                "properties" : {
                    ...
                    "password" : {
                        "title" : "Password",
                        ...
                        "encryption" : {
                            "purpose" : "idm.password.encryption"
                        },
                        "scope" : "private",
                    }
            ...
        }
    ]
}
```

The settings for reversible encryption depend on the encryption capabilities of the underlying JVM. refer to the explanations in javax.crypto.Cipher. You can accept the default settings, or specify the `cipher` and the `keySize`, for example:

```
...
    "encryption" : {
        "purpose": "idm.password.encryption",
        "cipher": "AES/GCM/NoPadding",
        "keySize": 128
},
```

The syntax for the `cipher` is algorithm/mode/padding, for example, `"cipher" : "AES/CBC/PKCS5Padding"`:

- The cipher algorithm defines how the plaintext is encrypted and decrypted.

  The default algorithm is the Advanced Encryption Standard (AES).

- The cipher mode defines how a block cipher algorithm transforms data larger than a single block.

  The default cipher mode is cipher block chaining (CBC).

- The cipher padding defines how to pad the plaintext to reach the appropriate size for the algorithm.

  The default cipher padding is PKCS#5 padding.

- The cipher key size determines the encryption strength, where longer key lengths strengthen encryption at the cost of lower performance.

  The default `keySize` is `16`.

> **Note**
>
> If you change the default cipher, you must specify the algorithm, mode, and padding. If the algorithm does not require a mode, use `NONE`. If the algorithm does not require padding, use `NoPadding`.

To encrypt attribute values from the command-line, refer to encrypt.

Configure encryption using the admin UI

1. Select **Configure > Managed Objects**, and select the object type whose property values you want to encrypt (for example, **User**).
2. On the **Properties** tab, select the property whose value should be encrypted and select the **Encrypt** checkbox.

Salted hash algorithms

To encode attribute values with salted hash algorithms, add the `secureHash` property to the attribute definition and define the hashing configuration. The configuration depends on the algorithm that you choose.

If you do not specify an algorithm, `SHA-256` is used by default. MD5 and SHA-1 are supported for legacy reasons, but should not be used in production environments.

Supported Hashing Algorithms and Configuration Properties

Algorithm	Config Property and Description
`BCRYPT`	**`cost`** Value between 4 and 31. Default is `13`.
`PBKDF2`	**`hashLength`** Byte-length of the generated hash. Default is `16`. **`saltLength`** Byte-length of the salt value. Default is `16`. **`iterations`** Number of cryptographic iterations. Default is `20000`. **`hmac`** HMAC algorithm. Default is `SHA3-256`. Supported values: • `SHA-224` • `SHA-256` • `SHA-384` • `SHA-512` • `SHA3-224` • `SHA3-256` • `SHA3-384` • `SHA3-512`

Algorithm	Config Property and Description
`SCRYPT`	**`hashLength`** Byte-length of the generated hash, must be greater than or equal to 8. Default is `16`. **`saltLength`** Byte-length of the salt value. Default is `16`. **`n`** CPU/Memory cost. Must be greater than 1, a power of 2, and less than *2^(128 * r / 8)*. Default is `32768`. **`p`** Parallelization. Must be a positive integer less than or equal to *Integer.MAX_VALUE / (128 * r * 8)*. Default is `1`. **`r`** Block size. Must be greater than or equal to 1. Default is `8`.
`SHA-256`	**`saltLength`** Byte-length of the salt value. Default is `16`. **Note** This is the default hashing.
`SHA-384`	**`saltLength`** Byte-length of the salt value. Default is `16`.
`SHA-512`	**`saltLength`** Byte-length of the salt value. Default is `16`.

The following list displays supported hash algorithms and example configurations:

SHA-256

```
"secureHash" : {
    "algorithm" : "SHA-256",
    "saltLength" : 16
}
```

SHA-384

```
"secureHash" : {
    "algorithm" : "SHA-384",
    "saltLength" : 16
}
```

SHA-512

```
"secureHash" : {
    "algorithm" : "SHA-512",
    "saltLength" : 16
}
```

Bcrypt

```
"secureHash" : {
    "algorithm" : "BCRYPT",
    "cost" : 16
}
```

Scrypt

```
"secureHash" : {
    "algorithm" : "SCRYPT",
    "hashLength" : 16,
    "saltLength" : 16,
    "n" : 32768,
    "r" : 8,
    "p" : 1
}
```

Password-Based Key Derivation Function 2 (PBKDF2)

```
"secureHash" : {
    "algorithm" : "PBKDF2",
    "hashLength" : 16,
    "saltLength" : 16,
    "iterations" : 10,
    "hmac" : "SHA-256"
}
```

Important

Some one-way hash functions are designed to be computationally *expensive*. Functions such as PBKDF2, Bcrypt, and Scrypt are designed to be relatively slow even on modern hardware. This makes them generally less susceptible to brute force attacks. *However*, computationally expensive functions can dramatically increase response times. If you use these functions, be aware of the performance impact and perform extensive testing before deploying your service in production. Do not use functions like PBKDF2 and Bcrypt for any accounts that are used for frequent, short-lived connections.

Hashing is a one-way operation, such that the original value cannot be recovered. Therefore, if you hash the value of any property, you cannot synchronize that property value to an external resource. For managed object properties with hashed values, you must either exclude those properties from the mapping or set a random default value if the external resource requires the property.

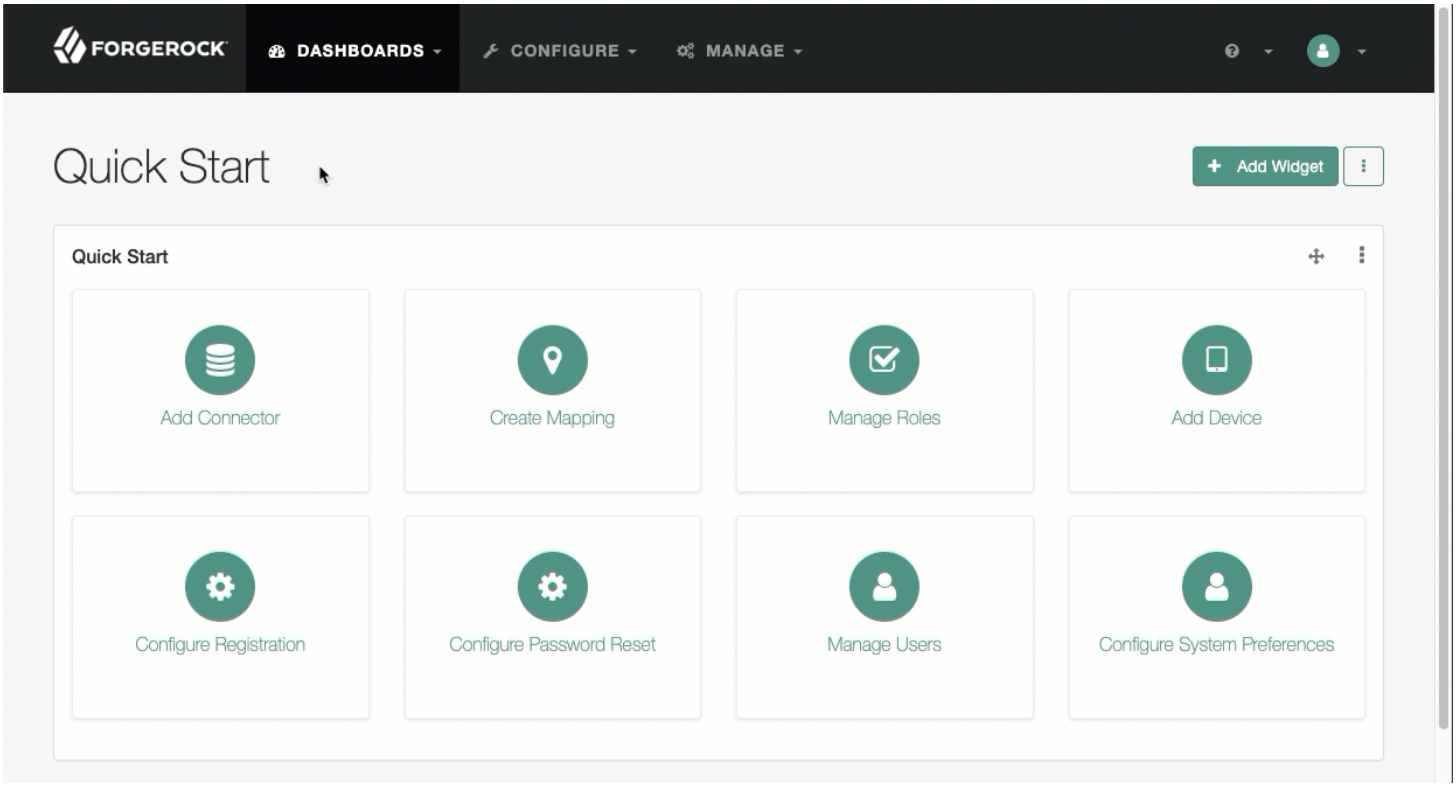
The following excerpt of a managed object configuration shows that values of the `password` attribute are hashed using the `SHA-256` algorithm:

```
{
    "objects" : [
        {
            "name" : "user",
            ...
            "schema" : {
                ...
                "properties" : {
                    ...
                    "password" : {
                        "title" : "Password",
                        ...
                        "secureHash" : {
                            "algorithm" : "SHA-256"
                        },
                        "scope" : "private",
                    }
            ...
        }
    ]
}
```

To hash attribute values from the command-line, refer to secureHash.

Configure hashing using the admin UI

You can set a property hash algorithm using the admin UI. However, only some algorithms and none of the enhanced configuration options are supported.



1. Select **Configure > Managed Objects**, and select an object type (for example, **User**).
2. On the **Properties** tab, select a property to hash.
3. On the **Property *Name*** page, click the **Privacy & Encryption** tab, and select **Hashed**.
4. From the adjacent drop-down menu, select an algorithm.
5. Click **Save**.

Encrypted objects

Encrypted objects and properties, such as passwords, include a `$crypto` object, that has the following structure:

```
"password": {
  "$crypto": {
    "type": "x-simple-encryption",
    "value": {
      "cipher": "AES/CBC/PKCS5Padding",
      "stableId": "openidm-sym-default",
      "salt": "Gwi+AGrn+VBOTmyq+TTuuw==",
      "data": "+9i7XAXpWZBXYTVEOBkM+w==",
      "keySize": 16,
      "purpose": "idm.password.encryption",
      "iv": "4xtI88eFu5tgfm8ooq+yqQ==",
      "mac": "N1zsYo71M/b/G6iLOhNohA=="
    }
  }
}
```

Most of the properties in the encrypted object `value` are self-explanatory and indicate how the property was encrypted. Specific IDM properties include the following:

- The `stableId` indicates the key alias that was used to encrypt the property value.
- The `purpose` refers to the secret ID used to encrypt the property value. For more information about secret IDs, refer to Secret stores.

Encrypt and decrypt properties over REST

The `openidm.encrypt` and `openidm.decrypt` functions of the Resource API enable you to encrypt and decrypt property values. To use these functions over the REST interface, run the `?_action=eval` action on the `script` endpoint.

The following example uses the `openidm.encrypt` function to encrypt a password value:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request POST \
--data '{
  "type": "text/javascript",
  "globals": {
    "val": {
      "myKey": "myPassword"
    }
  },
  "source":"openidm.encrypt(val,null,\"idm.password.encryption\");"
}' \
"https://localhost:8443/openidm/script?_action=eval"
{
  "$crypto": {
    "type": "x-simple-encryption",
    "value": {
      "cipher": "AES/CBC/PKCS5Padding",
      "stableId": "openidm-sym-default",
      "salt": "qAS/eG7zdnFyK5H8lXvqTA==",
      "data": "zewf6hR1yjp34EFJqUGpdnzzFCPJs2IaX4V97jdQlSI=",
      "keySize": 16,
      "purpose": "idm.password.encryption",
      "iv": "A4pIiY6kG6t0uLyLmJAoWQ==",
      "mac": "sFDJqg0Mmp0Ftl+1q1Bjzw=="
    }
  }
}
```

The following example uses the `openidm.decrypt` function to decrypt the password value:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request POST \
--data '{
  "type": "text/javascript",
  "globals": {
    "val": {
      "$crypto": {
        "type": "x-simple-encryption",
        "value": {
          "cipher": "AES/CBC/PKCS5Padding",
          "stableId": "openidm-sym-default",
          "salt": "qAS/eG7zdnFyK5H8lXvqTA==",
          "data": "zewf6hR1yjp34EFJqUGpdnzzFCPJs2IaX4V97jdQlSI=",
          "keySize": 16,
          "purpose": "idm.password.encryption",
          "iv": "A4pIiY6kG6t0uLyLmJAoWQ==",
          "mac": "sFDJqg0Mmp0Ftl+1q1Bjzw=="
        }
      }
    }
  },
  "source":"openidm.decrypt(val);"
}' \
"https://localhost:8443/openidm/script?_action=eval"
{
  "myKey": "myPassword"
}
```

For more information, refer to `openidm.encrypt` and `openidm.decrypt`.

Secure the repository

Configuration data and, in most deployments, user data, are stored in the IDM repository. In production deployments, you must secure access to the repository, and encrypt sensitive stored data.

For JDBC repositories, use a strong password for the connection to the repository and change at least the password of the database user (`openidm` by default). When you change the database username and/or password, update your database connection configuration file (`conf/datasource.jdbc-default.json`).

For a DS repository, change the `bindDN` and `bindPassword` for the directory server user in the `ldapConnectionFactories` property in the `repo.ds.json` file.

In both cases, the password is encrypted on server startup, using the key specified in the `idm.password.encryption` secret ID in `conf/secrets.json`.

Sensitive files and directories

Protect IDM files from access by unauthorized users. In particular, prevent other users from reading files in at least the `openidm/resolver/` and `openidm/security/` directories.

The objective is to limit access to the user that is running the service. Depending on the operating system and configuration, that user might be `root`, `Administrator`, `openidm`, or something similar.

Protect sensitive files in Unix

1. Make sure that user and group ownership of the installation and project directories is limited to the user running the IDM service.
2. Disable access of any sort for `other` users. One simple command for that purpose, from the `/path/to/openidm` directory, is:

```
chmod -R o-rwx .
```

Protect sensitive files in Windows

The IDM process in Windows is typically run by the `Local System` service account.

If you are concerned about the security of this account, you can set up a service account that only has permissions for IDM-related directories, then remove User access to the directories noted above. You should also configure the service account to deny local and remote login. For more information, refer to the User Rights Assignment article in Microsoft's documentation.

Adjust log levels

In production, set log levels to `INFO` to ensure that you capture enough information to help diagnose issues, but do not expose unnecessary information. For more information, refer to Server logs.

At startup and shutdown, `INFO` can produce many messages. During stable operation, `INFO` generally results in log messages only when coarse-grain operations such as scheduled reconciliation start or stop.

Secure the API Explorer

The REST API Explorer serves up interactive REST API documentation. The API Explorer can help you identify endpoints, and run REST calls against those endpoints. To protect production servers from unauthorized API descriptor requests, IDM requires authentication, by default. The property `authEnabled` protects static web resources from public view.

Default `ui.context-api.json` file

```
{
    "enabled" : true,
    "authEnabled" : true,
    "urlContextRoot" : "/api",
    "defaultDir" : "&{idm.install.dir}/ui/api/default",
    "extensionDir" : "&{idm.install.dir}/ui/api/extension"
}
```

To disable the API Explorer, set the following property in your `resolver/boot.properties` file:

```
openidm.apidescriptor.enabled=false
```

Hide unused REST endpoints

The two main use cases for IDM are data synchronization and user self-service.

If you are using IDM *only* to synchronize data sources, do not expose the server externally. In this case, IDM initiates all connections.

If you are using IDM *only* for user self-service, ensure that the server is behind a firewall or proxy, such as ForgeRock Identity Gateway. At a minimum, hide the `/admin` endpoint in the web interface via the proxy. Use the conf/access.json file as a guide for proxy or firewall rules.

If you are using IDM for data synchronization *and* user self-service, it is preferable to run two IDM servers or clusters, each with its own security model. Because the two use cases have very different load characteristics and security implications, running them on separate servers can help to prevent synchronization activity from impacting the performance on end-user systems.

Disable automatic configuration updates

By default, IDM monitors files in the `conf` directory periodically for any changes to the configuration. This functionality is configured in the following properties in your `conf/system.properties` file:

`openidm.fileinstall.poll`

Sets the interval at which files are polled for changes. `2000` milliseconds by default.

`openidm.fileinstall.enabled`

Specifies whether files should be monitored. `true` by default. In a production system, you should disable automatic polling for updates to prevent untested configuration changes from disrupting your identity service. Setting this property to `false` also disables the file-based configuration view, which means that IDM reads its configuration only from the repository.

> **Important**
>
> If automatic polling is enabled, IDM immediately uses changes to scripts called from a JSON configuration file.

`openidm.config.bootstrap.enabled`

Allows IDM to start up and load configuration when there aren't currently any configs stored in the repository. `true` by default.

`openidm.fileinstall.filter`

Specify which file types should be monitored (if `openidm.fileinstall.enabled=true`). File types are specified in a pipe-separated list, for example:

```
openidm.fileinstall.filter=.*\\.cfg|.*\\.json
```

Disable configuration file writes

To disable *writes* to configuration files, set the following property to `false` in your `conf/config.properties` file:

```
felix.fileinstall.enableConfigSave=false
```

This setting is suitable for read-only installations.

Secure IDM server files with a read-only installation

One method of locking down the server is to install IDM on a read-only file system. To accomplish this, complete all procedures in this topic.

This topic assumes that you have prepared the read-only volume appropriate for your Linux/UNIX installation environment and that you have set up a regular Linux user named `idm` and a dedicated volume for the `/idm` directory.

Prep

1. Configure the dedicated volume device, `/dev/volume` in the `/etc/fstab` file, as follows:

   ```
   /dev/volume/idm   ext4   ro,defaults   1,2
   ```

   When you run the `mount -a` command, the `/dev/volume` volume device is mounted on the `/idm` directory.

2. You can switch between read-write and read-only mode for the `/idm` volume with the following commands:

   ```
   sudo mount -o remount,rw /idm
   sudo mount -o remount,ro /idm
   ```

3. Confirm the result with the `mount` command, which should show that the `/idm` volume is mounted in read-only mode:

```
/dev/volumeon /idm type ext4 (ro)
```

4. Set up the `/idm` volume in read-write mode:

```
sudo mount -o remount,rw /idm
```

5. With the following commands, you can unpack the IDM binary in the `/idm` directory, and give user `idm` ownership of all files in that directory:

```
sudo unzip /idm/IDM-8.0.0.zip
sudo chown -R idm.idm /idm
```

Redirect audit and logging data

After you have installed IDM on a read-only file system, redirect audit and logging data to writable volumes. This procedure assumes a user `idm` with Linux administrative (superuser) privileges.

1. Create an external directory where IDM can send logging, auditing, and internal repository information:

```
sudo mkdir -p /var/log/openidm/audit
sudo mkdir /var/log/openidm/logs
sudo mkdir -p /var/cache/openidm/felix-cache
sudo mkdir /var/run/openidm
```

   Alternatively, route audit data to a remote data store. For an example of how to send audit data to a MySQL repository, refer to Direct audit information to MySQL.

2. Give the `idm` user ownership of the newly created directories:

```
sudo chown -R idm.idm /var/log/openidm
sudo chown -R idm.idm /var/cache/openidm
sudo chown -R idm.idm /var/run/openidm
```

3. Modify the following configuration files:

conf/audit.json

   Make sure the `handlerForQueries` is the JSON audit event handler and change the `logDirectory` property to the `/var/log/openidm/audit` subdirectory:

```
"eventHandlers" : [
    {
        "class" : "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
        "config" : {
            "name" : "json",
            "logDirectory" : "/var/log/openidm/audit",
            ...
        },
        ...
    }
]
```

conf/logging.properties

Change the `java.util.logging.FileHandler.pattern` property as follows:

```
java.util.logging.FileHandler.pattern = /var/log/openidm/logs/openidm%u.log
```

conf/config.properties

Activate and redirect the `org.osgi.framework.storage` property as follows:

```
# If this value is not absolute, then the felix.cache.rootdir controls
# how the absolute location is calculated. (See buildNext property)
org.osgi.framework.storage=&{felix.cache.rootdir|&{user.dir}}/felix-cache

# The following property is used to convert a relative bundle cache
# location into an absolute one by specifying the root to prepend to
# the relative cache path. The default for this property is the
# current working directory.
felix.cache.rootdir=/var/cache/openidm
```

Note

Your setup may require additional redirection for the following:

- Connectors. Depending on the connector, and the read-only volume, consider configuring connectors to direct output to writable volumes.
- Scripts. If you are using Groovy, examine the script configuration for your project. Make sure that output such as to the `groovy.target.directory` is directed to an appropriate location, such as `idm.data.dir` .

Finishing touches

1. Adjust the value of the `OPENIDM_PID_FILE` in the `startup.sh` and `shutdown.sh` scripts. To do so for a default bash shell, set the value of `OPENIDM_PID_FILE` for user `idm` by adding the following line to `/home/idm/.bashrc` :

```
export OPENIDM_PID_FILE=/var/run/openidm/openidm.pid
```

> **Note**
>
> For other shells, adjust your changes accordingly.

When you log in again as user `idm`, your `OPENIDM_PID_FILE` variable should redirect the process identifier file, `openidm.pid` to the `/var/run/openidm` directory, ready for access by the `shutdown.sh` script.

2. While the volume is still mounted in read-write mode, start IDM normally:

```
path/to/openidm/startup.sh -p project-dir
```

The first startup of IDM either processes the signed certificate that you added, or generates a self-signed certificate, and encrypts any passwords in the various configuration files.

3. Stop IDM.

4. You can now mount the `/idm` directory in read-only mode. The configuration in `/etc/fstab` ensures that Linux mounts the `/idm` directory in read-only mode on next system boot.

```
sudo mount -o remount,ro /idm
```

5. Reboot the system.

6. You can now start IDM, configured on a secure read-only volume.

```
path/to/openidm/startup.sh -p project-dir
```

Scripting

Guide to scripting for ForgeRock® Identity Management.

Scripting lets you extend IDM functionality. For example, you can provide custom logic between source and target mappings, define correlation rules, filters, triggers, and so on. This guide shows you how to use scripts in IDM and provides reference information on the script engine.

Script Configuration

Modify the parameters to compile, debug, and run scripts.

Custom Endpoints

Run arbitrary scripts through the REST URI.

Script Triggers

Learn where and how you can trigger scripts.

Script Variables

Learn about the variables available to scripts.

IDM supports scripts written in JavaScript and Groovy, and uses the following libraries:

- Rhino version 1.7.14 to run JavaScript.

> **Note**
>
> Rhino has limited support for ES6 / ES2015 (JavaScript version 1.7). For more information, refer to Rhino ES2015 Support.

- Groovy version 3.0.9 for Groovy script support.
- Lodash 3.10.1 and Handlebars 4.7.7 for Rhino scripting.

Note

Using Handlebars JS in server-side JS scripts requires *synchronization*; for example:

```
var Handlebars = require("lib/handlebars");
var result = new Packages.org.mozilla.javascript.Synchronizer(function() {
  var template = Handlebars.compile("Handlebars {{doesWhat}}");
  return template({ doesWhat: "rocks!" });
}, Handlebars)();
console.log(result);
```

- BouncyCastle 1.70 for signing JWTs.

Note

The BouncyCastle .JAR file that is bundled with IDM includes the `org.bouncycastle.asn1.util.Dump` command-line utility. Although this utility is not used directly by IDM, it is possible to reference the utility in your scripts. Due to a security vulnerability in this utility, you should *not* reference it in your scripts. For more information, refer to the corresponding BouncyCastle issue.

Important

Script options and locations are defined in the script configuration.
Default scripts are located in (`/path/to/openidm/bin/defaults/script/`). Do not modify the scripts in this directory. Rather copy the default scripts to a different location, make the changes, and update the referenced scripts in the applicable `conf/` file. You can put custom scripts in any of the locations referenced in the `sources` property in `conf/script.json` .

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

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

Script configuration

To modify the parameters used for compiling, debugging, and running scripts, edit the script configuration.

Script Configuration Parameters

properties

Any custom properties.

ECMAScript

JavaScript debug and compile options. JavaScript is an ECMAScript language.

- `javascript.optimization.level` - The current optimization level. Expected integer range is from -1 to 9. For more information about optimization level, refer to Rhino Optimization.

The default value is `9`.

- `javascript.recompile.minimumInterval` - The minimum time between script recompile.

  The default value is `60000`, or 60 seconds. This means that any changes made to scripts will not get picked up for up to 60 seconds. If you are developing scripts, reduce this parameter to around `100` (100 milliseconds).

  If you set the `javascript.recompile.minimumInterval` to `-1`, or remove this property from the script configuration, IDM does not poll JavaScript files to check for changes.

sources

The directories where IDM looks for referenced scripts.

Excerpt of a script configuration displaying default directories:

```
"sources" : {
   "default" : {
       "directory" : "&{idm.install.dir}/bin/defaults/script"
   },
   "install" : {
       "directory" : "&{idm.install.dir}"
   },
   "project" : {
       "directory" : "&{idm.instance.dir}"
   },
   "project-script" : {
       "directory" : "&{idm.instance.dir}/script"
}
```

> **Note**
>
> IDM loads scripts from `sources` in reverse order (bottom to top).

Call a script from the IDM configuration

To call a script from the IDM configuration, edit the configuration object. For example:

Provide a script source

```
{
    "type" : "text/javascript",
    "source": "scriptSource",
    "resourceBindings" : [{
        "resource" : "resourceName",
        "version" : "1.0",
        "binding" : "customName"
    }]
}
```

or

Provide a file reference

```
{
    "type" : "text/javascript",
    "file" : "file location"
}
```

Script variables are not necessarily simple `key:value` pairs, and can be any arbitrarily complex JSON object.

type

string, required

The script type.

IDM supports `"text/javascript"` and `"groovy"`.

source

string, required if `file` is not specified

Specifies the source code of the script to be executed.

resourceBindings

JSON object, optional

Allows specifying a resource, a vanity binding for that resource, and the API version the script should use. For example:

```
{
    "source" : "var response = consent.action(\"getConsentMappings\", {}); response[0];",
    "resourceBindings" : [{
        "resource" : "consent",
        "version" : "1.0",
        "binding" : "consent"
    }],
    "type" : "text/javascript"
}
```

This can improve the legibility of your scripts, by no longer needing to pass additional information within your script function.

file

string, required if `source` is not specified

Specifies the file containing the source code of the script to execute. The file path must be relative to project-dir. Absolute paths are not supported.

Tip

In general, you should namespace variables passed into scripts with the `globals` map. Passing variables in this way prevents collisions with the top-level reserved words for script maps, such as `source`, `file`, and `type`. This example uses the `globals` map to namespace the variables passed in the previous example.

```
"script": {
    "type" : "text/javascript",
    "file" : "script/triggerEmailNotification.js",
    "globals" : {
        "fromSender" : "admin@example.com",
        "toEmail" : "user@example.com"
    }
}
```

Examples

The following example script (in the mapping configuration) determines whether to include or ignore a target object in the reconciliation process based on an `employeeType` of `true`:

```
"validTarget" : {
    "type" : "text/javascript",
    "source" : "target.employeeType == 'external'"
}
```

The following example script (in the mapping configuration) sets the `__PASSWORD__` attribute to `defaultpwd` when IDM creates a target object:

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

Often, script files are reused in different contexts. You can pass variables to your scripts to provide these contextual details at runtime. You pass variables to the scripts that are referenced in configuration files by declaring the variable name in the script reference.

The following scheduled task configuration calls a script that triggers an email notification, but sets the sender and recipient of the email in the schedule configuration, rather than in the script itself:

```
{
    "enabled" : true,
    "type" : "cron",
    "schedule" : "0 0/1 * * * ?",
    "persisted" : true,
    "invokeService" : "script",
    "invokeContext" : {
        "script" : {
            "type" : "text/javascript",
            "file" : "script/triggerEmailNotification.js",
            "fromSender" : "admin@example.com",
            "toEmail" : "user@example.com"
        }
    }
}
```

Validate scripts over REST

IDM exposes a `script` endpoint over which scripts can be validated, by specifying the script parameters as part of the JSON payload. This functionality lets you test how a script will operate in your deployment, with complete control over the inputs and outputs. Testing scripts in this way can be useful in debugging.

The `script` endpoint supports two actions - `eval` and `compile`.

The `eval` action evaluates a script, by taking any actions referenced in the script, such as router calls to affect the state of an object. For JavaScript scripts, the last statement that is executed is the value produced by the script, and the expected result of the REST call.

The following REST call attempts to evaluate the `autoPurgeAuditRecon.js` script (provided in `openidm/bin/defaults/script/audit`), but provides an incorrect purge type (`"purgeByNumOfRecordsToKeep"` instead of `"purgeByNumOfReconsToKeep"`). The error is picked up in the evaluation. The example assumes that the script exists in the directory reserved for custom scripts (`openidm/script`):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "type": "text/javascript",
  "file": "script/autoPurgeAuditRecon.js",
  "globals": {
    "input": {
      "mappings": ["%"],
      "purgeType": "purgeByNumOfRecordsToKeep",
      "numOfRecons": 1
    }
  }
}' \
"http://localhost:8080/openidm/script?_action=eval"
"Must choose to either purge by expired or number of recons to keep"
```

Tip

The variables passed into this script are namespaced with the `globals` map. It is preferable to namespace variables passed into scripts in this way, to avoid collisions with the top-level reserved words for script maps, such as `file`, `source`, and `type`.

The `compile` action compiles a script, but does not execute it. A successful compilation returns `true`. An unsuccessful compilation returns the reason for the failure.

The following REST call tests whether a transformation script will compile:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "type":"text/javascript",
  "source":"source.mail ? source.mail.toLowerCase() : null"
}' \
"http://localhost:8080/openidm/script?_action=compile"
True
```

If the script is not valid, the action returns an indication of the error, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "type":"text/javascript",
  "source":"source.mail ? source.mail.toLowerCase()"
}' \
"http://localhost:8080/openidm/script?_action=compile"
{
  "code": 400,
  "reason": "Bad Request",
  "message": "missing : in conditional expression (386...BF2#1)in 386...BF2 at line number 1 at column
number 39"
}
```

Create custom endpoints to launch scripts

Custom endpoints let you run arbitrary scripts through the REST API.

Custom endpoint configuration

A custom endpoint configuration has the following structure:

```
{
    "context" : "context path",
    "type" : "script language",
    "source" : "script source" | "file" : "script file",
    "apiDescription" : "API descriptor object"
}
```

`context`

string, optional

The root URL path for the endpoint, in other words, the *route* to the endpoint. An endpoint with the context `endpoint/test` is addressable over REST at the URL `http://localhost:8080/openidm/endpoint/test` or by using a script such as `openidm.read("endpoint/test")`.

Endpoint contexts support wild cards, as shown in the preceding example. The `endpoint/linkedview/*` route matches the following patterns:

```
endpoint/linkedView/managed/user/bjensen
endpoint/linkedView/system/ldap/account/bjensen
endpoint/linkedView/
endpoint/linkedView
```

The `context` parameter is not mandatory in the endpoint configuration file. If you do not include a `context`, the route to the endpoint is identified by the name of the file. For example, in the sample endpoint configuration provided in `openidm/samples/example-configurations/custom-endpoint/conf/endpoint-echo.json`, the route to the endpoint is `endpoint/echo`.

`type`

string, required

The script type.

`file` *or* `source`

The path to the script file, or the script itself, inline.

For example:

```
"file" : "workflow/gettasksview.js"
```

or

```
"source" : "require('linkedView').fetch(request.resourcePath);"
```

Custom endpoint scripts

The custom endpoint script files in the `samples/example-configurations/custom-endpoint/script` directory demonstrate all the HTTP operations that can be called by a script. Each HTTP operation is associated with a `method` (`create`, `read`, `update`, `delete`, `patch`, `action`, or `query`). Requests sent to the custom endpoint return a list of the variables available to each method.

All scripts are invoked with a global `request` variable in their scope. This request structure carries all the information about the request.

> ⚠ **Warning**
>
> Read requests on custom endpoints must not modify the state of the resource, either on the client or the server, as this can make them susceptible to CSRF exploits.
> The standard READ endpoints are safe from Cross Site Request Forgery (CSRF) exploits because they are inherently read-only. That is consistent with the *Guidelines for Implementation of REST*, from the US National Security Agency, as "... CSRF protections need only be applied to endpoints that will modify information in some way."

Custom endpoint scripts *must* return a JSON object. The structure of the return object depends on the `method` in the request.

The following example shows the `create` method in the `echo.js` file:

```
if (request.method === "create") {
    return {
        method: "create",
        resourceName: request.resourcePath,
        newResourceId: request.newResourceId,
        parameters: request.additionalParameters,
        content: request.content,
        context: context.current
    }
}
```

Depending on the method, the variables available to the script can include the following:

`resourceName`

The name of the resource, without the `endpoint/` prefix, such as `echo`.

`newResourceId`

The identifier of the new object, available as the results of a `create` request.

`revision`

The revision of the object.

`parameters`

Any additional parameters provided in the request. The sample code returns request parameters from an HTTP GET with `?param=x`, as `"parameters":{"param":"x"}`.

`content`

Content based on the latest revision of the object, using `getObject`.

`context`

The context of the request, including headers and security. For more information, refer to Request context chain.

Paging parameters

The `pagedResultsCookie`, `pagedResultsOffset`, and `pageSize` parameters are specific to `query` methods. For more information refer to Page Query Results.

Query parameters

The `queryId` and `queryFilter` parameters are specific to `query` methods. For more information refer to Construct Queries.

Script exceptions

Some custom endpoint scripts require exception-handling logic. To return meaningful messages in REST responses and in logs, you must comply with the language-specific method of throwing errors.

A script written in JavaScript should comply with the following exception format:

```
throw {
    "code": 400, // any valid HTTP error code
    "message": "custom error message",
    "detail" : {
        "var": parameter1,
        "complexDetailObject" : [
            "detail1",
            "detail2"
        ]
    }
}
```

Any exceptions will include the specified HTTP error code, the corresponding HTTP error message, such as `Bad Request`, a custom error message that can help you diagnose the error, and any additional detail that you think might be helpful.

Register custom scripted actions

You can register custom scripts that initiate some arbitrary action on a managed object endpoint. You can declare any number of actions in your managed object schema and associate those actions with a script.

The return value of a custom scripted action is ignored. The managed object is returned as the response of the scripted action, whether that object has been updated by the script or not.

Custom scripted actions have access to the following variables:

- `context`
- `request`
- `resourcePath`
- `object`

Example scenario

In this scenario, you want your managed users to have the option to receive update notifications. You can define an *action* that toggles the value of a specific property on the user object.

1. Add an `updates` property to the managed object configuration:

```
"properties": {
    ...
    "updates": {
        "title": "Automatic Updates",
        "viewable": true,
        "type": "boolean",
        "searchable": true,
        "userEditable": true
    },
    ...
}
```

2. Add a `toggleUpdates` action to the managed user object definition:

```
{
    "objects" : [
        {
            "name" : "user",
            "onCreate" : {
                ...
            },
            ...
            "actions" : {
                "toggleUpdates" : {
                    "type" : "text/javascript",
                    "source" : "openidm.patch(resourcePath, null, [{ 'operation' : 'replace', 'field' : '/
updates', 'value' : !object.updates }])"
                }
            },
            ...
        }
    ]
}
```

> **Note**
>
> The `toggleUpdates` action calls a script that changes the value of the user's `updates` property.

3. To call the script, specify the ID of the action in a POST request on the user object:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/managed/user/ID?_action=toggleUpdates"
```

You can now test the functionality.

4. Create a managed user, `bjensen`, with an `updates` property set to `true`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "userName":"bjensen",
  "sn":"Jensen",
  "givenName":"Barbara",
  "mail":"bjensen@example.com",
  "telephoneNumber":"5556787",
  "description":"Created by OpenIDM REST.",
  "updates": true,
  "password":"Passw0rd"
}' \
"http://localhost:8080/openidm/managed/user?_action=create"
{
  "_id": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
  "_rev": "0000000050c62938",
  "userName": "bjensen",
  "sn": "Jensen",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "telephoneNumber": "5556787",
  "description": "Created by OpenIDM REST.",
  "updates": true,
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

5. Run the `toggleUpdates` action on `bjensen`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/managed/user/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb?
_action=toggleUpdates"
{
  "_id": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
  "_rev": "00000000a92657c7",
  "userName": "bjensen",
  "sn": "Jensen",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "telephoneNumber": "5556787",
  "description": "Created by OpenIDM REST.",
  "updates": false,
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

Note

Note in the command output that this action has set bjensen's `updates` property to `false`.

Request context chain

The context chain of any request is established as follows:

1. The request starts with a *root context*, associated with a specific context ID.
2. The root context is wrapped in the *security context* that includes the authentication and authorization detail for the request.
3. The security context is further wrapped by the *HTTP context*, with the target URI. The HTTP context is associated with the normal parameters of the request, including a user agent, authorization token, and method.
4. The HTTP context is wrapped by one or more server/router context(s), with an endpoint URI. The request can have several layers of server and router contexts.

Script triggers

Tip

For more information about the variables available to scripts, refer to Script variables.

Scripts can be triggered in different places, and by different events. The following list indicates the configuration files in which scripts can be referenced, the events upon which the scripts can be triggered, and the actual scripts that can be triggered on each of these files.

Scripts called in mappings

Triggered by situation

onCreate, onUpdate, onDelete, onLink, onUnlink

Object filter

validSource, validTarget

Triggered when correlating objects

correlationQuery, correlationScript

Triggered on any reconciliation

result

Scripts inside properties

condition, transform

`sync.json` supports only one script per hook. If multiple scripts are defined for the same hook, only the last one is kept.

Scripts inside policies

condition

Within a synchronization policy, you can use a `condition` script to apply different policies based on the link type, for example:

```
"condition" : {
  "type" : "text/javascript",
  "source" : "linkQualifier == \"user\""
}
```

Scripts called in the managed object configuration

onCreate, onRead, onUpdate, onDelete, onValidate, onRetrieve, onStore, onSync, postCreate, postUpdate, and postDelete

The managed object configuration supports only one script per hook. If multiple scripts are defined for the same hook, only the last one is kept.

Scripts called in the router configuration

onRequest, onResponse, onFailure

The router configuration supports multiple scripts per hook.

managed object configuration">

Script triggers defined in the managed object configuration

For information about how managed objects are handled and the available script triggers, refer to Managed objects reference.

Managed object configuration object

Trigger	Variable
`onCreate`, `postCreate`	• **object**: The content of the object being created. • **newObject**: The object after the create operation is complete. • **context**: Information related to the current request, such as client, end user, and routing. • **resourceName**: The resource path of the object of the query. For example, if you create a managed user with ID `42f8a60e-2019-4110-a10d-7231c3578e2b`, resourceName returns `managed/user/42f8a60e-2019-4110-a10d-7231c3578e2b`. • **request**: Information related to the request, such as headers, credentials, and the desired action. Also includes the endpoint, and payload to be processed.
`onUpdate`, `postUpdate` *Returns JSON object*	• **object**: The content of the object being updated. • **oldObject**: The state of the object, before the update. • **newObject**: Changes to be applied to the object. May continue with the `onUpdate` trigger. • **context**: Information related to the current request, such as client, end user, and routing. • **resourceName**: The resource path of the object the query. • **request**: Information related to the request, such as headers, credentials, and the desired action. Also includes the endpoint, and payload to be processed.
`onDelete`, `onRetrieve`, `onRead` *Returns JSON object.*	• **object**: The content of the object. • **context**: Information related to the current request, such as client, end user, and routing. • **resourceName**: The resource path of the object the query. • **request**: Information related to the request, such as headers, credentials, and the desired action. Also includes the endpoint, and payload to be processed.

Trigger	Variable
`postDelete` *Returns JSON object.*	• **oldObject**: Represents the deleted object. • **context**: Information related to the current request, such as client, end user, and routing. • **resourceName**: The resource path of the object the query is performed upon. • **request**: Information related to the request, such as headers, credentials, and the desired action. Also includes the endpoint, and payload to be processed.

`onSync`
Returns JSON object
	• **oldObject**: Represents the object prior to sync. If sync has not been run before, the value will be `null`.
	• **newObject**: Represents the object after sync is completed.
	• **context**: Information related to the current request, such as client, end user, and routing.
	• **request**: Information related to the request, such as headers, credentials, and the desired action. Also includes the endpoint, and payload to be processed.
	• **resourceName**: An object representing the resource path the query is performed upon.
	• **syncResults**: A map containing the results and details of the sync, including:
	  ◦ **success** (boolean): Success or failure of the sync operation.
	  ◦ **action**: Returns the name of the action performed as a string.
	  ◦ **syncDetails**: The mappings attempted during synchronization.

`onStore`, `onValidate` *Returns JSON object*	• **object**: Represents the object being stored or validated. • **value**: The content to be stored or validated for the object. • **context**: Information related to the current request, such as client, end user, and routing. • **resourceName**: The resource path of the object the query is performed upon. • **request**: Information related to the request, such as headers, credentials, and the desired action. Also includes the endpoint, and payload to be processed.

Property object

Trigger	Variable
`onRetrieve`, `onStore` *Returns JSON object*	• **object**: Represents the object being operated upon. • **property**: The value of the property being retrieved or stored. • **propertyName**: The name of the property being retrieved or stored. • **context**: Information related to the current request, such as client, end user, and routing. • **resourceName**: The resource path of the object the query is performed upon. • **request**: Information related to the request, such as headers, credentials, and the desired action. Also includes the endpoint, and payload to be processed.
`onValidate` *Returns JSON object*	• **property**: The value of the property being validated. • **context**: Information related to the current request, such as client, end user, and routing. • **resourceName**: The resource path of the object the query is performed upon. • **request**: Information related to the request, such as headers, credentials, and the desired action. Also includes the endpoint, and payload to be processed.

Script triggers defined in mappings

For information about how managed objects in mappings are handled, and the script triggers available, refer to Object-Mapping Objects.

Object-mapping object

Trigger	Variable
`correlationQuery`, `correlationScript` *Returns JSON object*	• **source**: Represents the source object. • **linkQualifier**: The link qualifier associated with the current sync. • **context**: Information related to the current request, such as source and target.

Trigger	Variable
`linkQualifiers` *Returns JSON object*	• **mapping**: The name of the current mapping. • **object**: The value of the source object. During a DELETE event, that source object may not exist, and may be null. • **oldValue**: The former value of the deleted source object, if any. If the source object is new, oldValue will be null. When there are deleted objects, oldValue is populated only if the source is a managed object. • **returnAll** (boolean): Link qualifier scripts must return every valid link qualifier when returnAll is true, independent of the source object. If returnAll is true, the script must not attempt to use the object variable, because it will be null. It's best practice to configure scripts to start with a check for the value of returnAll. • **context**: Information related to the current request, such as source and target.
`onCreate` *Returns JSON object*	• **source**: Represents the source object. • **target**: Represents the target object. • **situation**: The situation associated with the current sync operation. • **linkQualifier**: The link qualifier associated with the current sync operation. • **context**: Information related to the current sync operation. • **sourceId**: The object ID for the source object. • **targetId**: The object ID for the target object. • **mappingConfig**: A configuration object representing the mapping being processed.

Trigger	Variable
`onDelete`, `onUpdate` *Returns JSON object*	• **source**: Represents the source object. • **target**: Represents the target object. • **oldTarget**: Represents the target object prior to the DELETE or UPDATE action. • **situation**: The situation associated with the current sync operation. • **linkQualifier**: The link qualifier associated with the current sync. • **context**: Information related to the current sync operation. • **sourceId**: The object ID for the source object. • **targetId**: The object ID for the target object. • **mappingConfig**: A configuration object representing the mapping being processed.
`onLink`, `onUnlink` *Returns JSON object*	• **source**: Represents the source object. • **target**: Represents the target object. • **linkQualifier**: The link qualifier associated with the current sync operation. • **context**: Information related to the current sync operation. • **sourceId**: The object ID for the source object. • **targetId**: The object ID for the target object. • **mappingConfig**: A configuration object representing the mapping being processed.

Trigger	Variable
`onError` *Returns JSON object*	• **source**: Represents the source object. • **target**: Represents the target object. • **linkQualifier**: The link qualifier associated with the current sync operation. • **context**: Information related to the current sync operation. • **situation**: The situation associated with the current sync operation. • **sourceId**: The object ID for the source object. • **targetId**: The object ID for the target object. • **oldSource**: Available during UPDATE and DELETE operations performed through implicit sync. With implicit synchronization, the synchronization operation is triggered by a specific change to the source object. As such, implicit sync can populate the old value within the `oldSource` variable and pass it on to the sync engine. • **error**: The result of the resource exception, as a JSON object. • **mappingConfig**: A configuration object representing the mapping being processed.
`result` *Returns JSON object of reconciliation results*	• **source**: Provides statistics about the source phase of the reconciliation. • **target**: Provides statistics about the target phase of the reconciliation. • **context**: Information related to the current operation, such as source and target. • **global**: Provides statistics about the entire reconciliation operation. • **mappingConfig**: A configuration object representing the mapping being processed. • **reconState**: Provides the state of reconciliation operation; such as, *success*, *failure*, or *active*.
`validSource` *Returns boolean*	• **source**: Represents the source object. • **linkQualifier**: The link qualifier associated with the current sync operation. • **context**: Information related to the current sync operation.

Trigger	Variable
`validTarget` *Returns boolean*	• **target**: Represents the target object. • **linkQualifier**: The link qualifier associated with the current sync operation. • **context**: Information related to the current sync operation.

Property object

Trigger	Variable
`condition` *Returns boolean*	• **object**: The current object being mapped. • **context**: Information related to the current operation, such as source and target. • **linkQualifier**: The link qualifier associated with the current sync operation. • **target**: Represents the target object. • **oldTarget**: Represents the target object prior to any changes. • **oldSource**: Available during UPDATE and DELETE operations performed through implicit sync. With implicit synchronization, the synchronization operation is triggered by a specific change to the source object. As such, implicit sync can populate the old value within the `oldSource` variable and pass it on to the sync engine. During reconciliation operations `oldSource` will be undefined. A reconciliation operation cannot populate the value of the `oldSource` variable as it has no awareness of the specific change to the source object. Reconciliation simply synchronizes the static source object to the target.
`transform` *Returns JSON object*	• **source**: Represents the source object. • **linkQualifier**: The link qualifier associated with the current sync operation. • **context**: Information related to the current sync operation.

Policy object

Trigger	Variable
`action` *Returns string OR JSON object*	• **source**: Represents the source object. • **target**: Represents the target object. • **sourceAction** (boolean): Indicates whether the action is being processed during the source or target synchronization phase (true if performed during a source synchronization, false if performed during a target synchronization). • **linkQualifier**: The link qualifier used for this operation (`default` if no other link qualifier is specified). • **context**: Information related to the current sync operation. • **recon**: Represents the reconciliation operation. • The `recon.actionParam` object contains information about the current reconciliation operation and includes the following variables: ◦ `reconId` : The ID of the reconciliation operation. ◦ `mapping` : The mapping for which the reconciliation was performed, for example, `systemLdapAccounts_managedUser` . ◦ `situation` : The situation encountered, for example, AMBIGUOUS. ◦ `action` : The default action that would be used for this situation, if not for this script. The script being executed replaces the default action (and is used instead of any other named action). ◦ `sourceId` : The `_id` value of the source record. ◦ `linkQualifier` : The link qualifier used for that mapping, (`default` if no other link qualifier is specified). ◦ `ambiguousTargetIds` : An array of the target object IDs that were found in an AMBIGUOUS situation during correlation. ◦ `_action` : The synchronization action (only `performAction` is supported).

Trigger	Variable
`postAction` *Returns JSON object*	• **source**: Represents the source object. • **target**: Represents the target object. • **action**: The sync action that was performed. • **sourceAction** (boolean): Indicates whether the action is being processed during the source or target synchronization phase (true if performed during a source synchronization, false if performed during a target synchronization). • **linkQualifier**: The link qualifier used for this operation (`default` if no other link qualifier is specified). • **reconId**: Represents the ID of the reconciliation. • **situation**: Represents the situation for this policy. • **context**: Information related to the current operation, such as source and target.

router configuration">

Script triggers defined in the router configuration

Trigger	Variable
`onFailure`	exception
`onRequest`	request
`onResponse`	response

The `augmentSecurityContext` trigger

The `augmentSecurityContext` trigger, defined in the authentication configuration, can reference a script that is executed after successful authentication. Such scripts can populate the security context of the authenticated user. If the authenticated user is not found in the resource specified by `queryOnResource`, the `augmentSecurityContext` can provide the required authorization map.

Such scripts have access to the following bindings:

- `security` - includes the `authenticationId` and the `authorization` key, which includes the `moduleId`.

The main purpose of an `augmentSecurityContext` script is to modify the `authorization` map that is part of this `security` binding. The authentication module determines the value of the `authenticationId`, and IDM attempts to populate the `authorization` map with the details that it finds, related to that `authenticationId` value. These details include the following:

- `security.authorization.component` - the resource that contains the account (this will always will be the same as the value of `queryOnResource` by default).
- `security.authorization.id` - the internal `_id` value that is associated with the account.
- `security.authorization.roles` - any roles that were determined, either from reading the `userRoles` property of the account or from calculation.
- `security.authorization.moduleId` - the authentication module responsible for performing the original authentication.

You can use the `augmentSecurityContext` script to change any of these `authorization` values. The script can also add new values to the `authorization` map, which will be available for the lifetime of the session.

- `properties` - corresponds to the `properties` map of the related authentication module.
- `httpRequest` - a reference to the `Request` object that was responsible for handling the incoming HTTP request.

  This binding is useful to the augment script because it has access to all of the raw details from the HTTP request, such as the headers. The following code snippet shows how you can access a header using the `httpRequest` binding. This example accesses the `authToken` request header:

  ```
  httpRequest.getHeaders().getFirst('authToken').toString()
  ```

Script variables

> **Tip**
>
> For more information about the variables available in script triggers, refer to Script triggers.

The variables available to a script depend on several factors:

- The trigger that launches the script.
- The configuration file in which that trigger is defined.
- The object type:
  - For objects defined in the managed object configuration, the object type is either a managed object, or a managed object property.
  - For objects defined in the mapping configuration, the object can be an object-mapping object, a property object, or a policy object. For more information, refer to Policy Objects).

The following subtopics list the variables available to scripts, based on the configuration file in which the trigger is defined.

Variables available to scripts in custom endpoints

All custom endpoint scripts have a `request` variable in their scope, which is a JSON object containing all information about the request. The parameters found in this object vary depending on the request method. The request may include headers, credentials, and the desired action. The request normally also includes the endpoint as well as the payload to be processed.

For more details about writing custom endpoint scripts, refer to Custom Endpoint Scripts.

Variable	Variable Parameters
`request`	• **method**: The type of request, such as `query`, `create`, or `delete`. • **resourceName**: The name of the resource associated with the request. • **revision**: The revision number of the requested object. • **parameters**: JSON object mapping any additional parameters sent in the request. • **content**: The contents of the requested object. • **context**: Information related to the current request, such as client, end user, and routing. **Only available in `query` requests** • **pagedResultsCookie**: Represents the cookie used for `queryFilter` operations to track the results of a filtered query. • **pagedResultsOffset**: Specifies how many records to skip before returning a set of results. • **pageSize**: Specifies how many results to return per page. • **queryExpression**: A string containing a native query to query a system resource directly. • **queryId**: A string using the id of a predefined query object to return a specific set of results from a queried object. • **queryFilter**: A string with a common expression used to filter the results of a queried object. **Only available in `create` requests** • **newResourceId**: The ID of the new object. Only available in `create` requests.

Variables available to role assignment scripts

The optional `onAssignment` and `onUnassignment` event scripts specify what should happen to attributes that are affected by role assignments when those assignments are applied to a user, or removed from a user.

These scripts have access to the following variables:

- `sourceObject`
- `targetObject`
- `existingTargetObject`
- `linkQualifier`

The standard assignment scripts, `replaceTarget.js`, `mergeWithTarget.js`, `removeFromTarget.js`, and `noOp.js` have access to all the variables in the previous list, as well as the following:

- `attributeName`
- `attributeValue`
- `attributesInfo`

> **Note**
>
> Role assignment scripts must always return `targetObject`, otherwise other scripts and code that occur downstream of your script will not work as expected.

The `identityServer` variable

IDM provides an additional variable, named `identityServer`, to scripts. You can use this variable in several ways. The `ScriptRegistryService`, described in Validate scripts over REST, binds this variable to:

- `getProperty`

  Retrieves property information from system configuration files. Takes up to three parameters:

  - The name of the property you are requesting.
  - *(Optional)* The default result to return if the property wasn't set.
  - *(Optional)* Boolean to determine whether or not to use property substitution when getting the property.

  For example, you can retrieve the value of the `openidm.config.crypto.alias` property with the following code:
  `alias = identityServer.getProperty("openidm.config.crypto.alias", "true", true);`

- `getInstallLocation`

  Retrieves the IDM installation path, such as `/path/to/openidm`. May be superseded by an absolute path.

Router configuration

The router service provides the uniform interface to all IDM objects: managed objects, system objects, configuration objects, and so on.

The router configuration contains an array of Filter objects:

```
{
  "filters": [ filter object, ... ]
}
```

Filter objects

The required filters array defines a list of filters to be processed on each router request. Filters are processed in the order in which they are specified in this array, and have the following configuration:

```
{
  "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. Pattern matching is done on the resource name, rather than on individual objects.

methods

array of strings, optional

One or more methods for which the script(s) should be triggered. Supported methods are: `"create"`, `"read"`, `"update"`, `"delete"`, `"patch"`, `"query"`, `"action"`. If not specified, all methods are matched.

condition

script object, optional

Specifies a script that is called first to determine if the script should be triggered. If the condition yields `"true"`, the other script(s) are executed. If no condition is specified, the script(s) are called unconditionally.

onRequest

script object, optional

Specifies a script to execute before the request is dispatched to the resource. If the script throws an exception, the method is not performed, and a client error response is provided.

onResponse

script object, optional

Specifies a script to execute after the request is successfully dispatched to the resource and a response is returned. Throwing an exception from this script does not undo the method already performed.

onFailure

script object, optional

Specifies a script to execute if the request resulted in an exception being thrown. Throwing an exception from this script does not undo the method already performed.

router configuration">

Pattern matching in the router configuration

Pattern matching can minimize overhead in the router service. The default router configuration includes instances of the `pattern` filter object, that limit script requests to specified methods and endpoints.

Based on the following code snippet, the router service would trigger the `policyFilter.js` script for `CREATE` and `UPDATE` calls to managed and internal objects:

```
{
    "pattern" : "^(managed|internal)($|(/.+))",
    "onRequest" : {
        "type" : "text/javascript",
        "source" : "require('policyFilter').runFilter()"
    },
    "methods" : [
        "create",
        "update"
    ]
}
```

Without this `pattern`, IDM would apply the policy filter to additional objects, such as the audit service, which could affect performance.

Script execution sequence

All `onRequest` and `onResponse` scripts are executed in sequence. First, the `onRequest` scripts are executed from the top down, then the `onResponse` scripts are executed from the bottom up.

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

The following sample router configuration shows the order in which the scripts would be executed:

```
{
    "filters" : [
        {
            "onRequest" : {
                "type" : "text/javascript",
                "source" : "require('router-authz').testAccess()"
            }
        },
        {
            "pattern" : "^managed/user",
            "methods" : [
                "read"
            ],
            "onRequest" : {
                "type" : "text/javascript",
                "source" : "console.log('requestFilter 1');"
            }
        },
        {
            "pattern" : "^managed/user",
            "methods" : [
                "read"
            ],
            "onResponse" : {
                "type" : "text/javascript",
                "source" : "console.log('responseFilter 1');"
            }
        },
        {
            "pattern" : "^managed/user",
            "methods" : [
                "read"
            ],
            "onRequest" : {
                "type" : "text/javascript",
                "source" : "console.log('requestFilter 2');"
            }
        },
        {
            "pattern" : "^managed/user",
            "methods" : [
                "read"
            ],
            "onResponse" : {
                "type" : "text/javascript",
                "source" : "console.log('responseFilter 2');"
            }
        }
    ]
}
```

This configuration would produce a log as follows:

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

This 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"
            }
        }
    ]
}
```

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 IDM resources.

request

resource-request object

The resource-request context, which has one or more parent contexts. Provided in the scope of all scripts. For more information about the request context, refer to Request context chain.

response

resource-response 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 `"onFailure"` script. An exception object is defined as:

```
{
  "code": integer,
  "reason": string,
  "message": string,
  "detail": string
}
```

code

integer

The numeric HTTP code of the exception.

reason

string

The short reason phrase of the exception.

message

string

A brief message describing the exception.

detail

(optional), string

A detailed description of the exception, in structured JSON format, suitable for programmatic evaluation.

Scripting function reference

> **Note**
>
> If you need to request specific resource versions, refer to REST API versioning.

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

Note

Most of the following function examples are in JavaScript. To use the functions in Groovy scripts, make adjustments as necessary. For example, you need to pass parameters using square brackets (not curly braces):

```
openidm.query("managed/user", ["_queryFilter" : "/userName sw \"user.1\""], ["userName", "_id"])
```

The script engine supports the following functions:

This function creates a new resource object.

Parameters

resourceName

string

The container in which the object will be created, for example, `managed/user`.

newResourceId

string

The identifier of the object to be created, if the client is supplying the ID. If the server should generate the ID, pass null here.

content

JSON object

The content of the object to be created.

params

JSON object (optional)

Additional parameters that are passed to the create request.

fields

JSON array (optional)

An array of the fields that should be returned in the result. The list of fields can include wild cards, such as `*` or `*_ref`. If no fields are specified, the entire new object is returned.

Returns

The created resource object.

Throws

An exception is thrown if the object could not be created.

Example

```
openidm.create("managed/user", ID, JSON object);
```

This function performs a partial modification of a managed or system object. Unlike the `update` function, only the modified attributes are provided, not the entire object.

Parameters

resourceName

string

The full path to the object being updated, including the ID.

rev

string

The revision of the object to be updated. Use `null` if the object is not subject to revision control, or if you want to skip the revision check and update the object, regardless of the revision.

value

An array of one or more JSON objects

The value of the modifications to be applied to the object. The patch set includes the operation type, the field to be changed, and the new values. A PATCH request can `add`, `remove`, `replace`, or `increment` an attribute value.

A `remove` operation removes a property if the value of that property equals the specified value, or if no value is specified in the request. The following example `value` removes the `marital_status` property from the object, *if* the value of that property is `single`:

```
[
    {
        "operation": "remove",
        "field": "marital_status",
        "value": "single"
    }
]
```

For fields whose value is an array, it's not necessary to know the position of the value in the array, as long as you specify the full object. If the full object is found in the array, that value is removed. The following example removes user adonnelly from bjensen's `reports`:

```
{
    "operation": "remove",
    "field": "/manager",
    "value": {
      "_ref": "managed/user/adonnelly",
      "_refResourceCollection": "managed/user",
      "_refResourceId": "adonnelly",
      "_refProperties": {
        "_id": "ed6620e4-98ba-410c-abc0-e06dc1be7aa7",
        "_rev": "000000008815942b"
      }
    }
}
```

If an invalid value is specified (that is a value that does not exist for that property in the current object) the patch request is silently ignored.

A `replace` operation replaces an existing value, or adds a value if no value exists.

params

JSON object (optional)

Additional parameters that are passed to the patch request.

fields

JSON array (optional)

An array of the fields that should be returned in the result. The list of fields can include wild cards, such as `*` or `*_ref`. If no fields are specified, the entire new object is returned.

Returns

The modified resource object.

Throws

An exception is thrown if the object could not be updated.

Examples

Patching an object to add a value to an array:

```
openidm.patch("managed/role/" + role._id, null, [{"operation":"add", "field":"/members/-", "value":
{"_ref":"managed/user/" + user._id}}]);
```

Patching an object to remove an existing property:

```
openidm.patch("managed/user/" + user._id, null, [{"operation":"remove", "field":"marital_status",
"value":"single"}]);
```

Patching an object to replace a field value:

```
openidm.patch("managed/user/" + user._id, null, [{"operation":"replace", "field":"/password",
"value":"Passw0rd"}]);
```

Patching an object to increment an integer value:

```
openidm.patch("managed/user/" + user._id, null, [{"operation":"increment","field":"/age","value":1}]);
```

This function reads and returns a resource object.

Parameters

resourceName

string

The full path to the object to be read, including the ID.

params

JSON object (optional)

The parameters that are passed to the read request. Generally, no additional parameters are passed to a read request, but this might differ, depending on the request. If you need to specify a list of `fields` as a third parameter, and you have no additional `params` to pass, you must pass `null` here. Otherwise, you simply omit both parameters.

fields

JSON array (optional)

An array of the fields that should be returned in the result. The list of fields can include wild cards, such as `*` or `*_ref`. If no fields are specified, the entire object is returned.

Returns

The resource object, or `null` if not found.

Example

```
openidm.read("managed/user/"+userId, null, ["*", "manager"]);
```

This function updates an entire resource object.

Parameters

id

string

The complete path to the object to be updated, including its ID.

rev

string

The revision of the object to be updated. Use `null` if the object is not subject to revision control, or if you want to skip the revision check and update the object, regardless of the revision.

value

object

The complete replacement object.

params

JSON object (optional)

The parameters that are passed to the update request.

fields

JSON array (optional)

An array of the fields that should be returned in the result. The list of fields can include wild cards, such as `*` or `*_ref`. If no fields are specified, the entire object is returned.

Returns

The modified resource object.

Throws

An exception is thrown if the object could not be updated.

Example

In this example, the managed user entry is read (with an `openidm.read`, the user entry that has been read is updated with a new description, and the entire updated object is replaced with the new value.

```
var user_read = openidm.read('managed/user/' + source._id);
user_read['description'] = 'The entry has been updated';
openidm.update('managed/user/' + source._id, null, user_read);
```

This function deletes a resource object.

Parameters

resourceName

string

The complete path to the to be deleted, including its ID.

rev

string

The revision of the object to be deleted. Use `null` if the object is not subject to revision control, or if you want to skip the revision check and delete the object, regardless of the revision.

params

JSON object (optional)

The parameters that are passed to the delete request.

fields

JSON array (optional)

An array of the fields that should be returned in the result. The list of fields can include wild cards, such as `*` or `*_ref`. If no fields are specified, the entire object is returned.

Returns

Returns the deleted object if successful.

Throws

An exception is thrown if the object could not be deleted.

Example

```
openidm.delete('managed/user/'+ user._id, user._rev);
```

This function performs a query on the specified resource object. For more information, refer to Construct Queries.

Parameters

resourceName

string

The resource object on which the query should be performed, for example, `"managed/user"`, or `"system/ldap/account"`.

params

JSON object

The parameters that are passed to the query (`_queryFilter`, or `_queryId`). Additional parameters passed to the query will differ, depending on the query.

Certain common parameters can be passed to the query to restrict the query results. The following sample query passes paging parameters and sort keys to the query.

```
reconAudit = openidm.query("audit/recon", {
    "_queryFilter": queryFilter,
    "_pageSize": limit,
    "_pagedResultsOffset": offset,
    "_pagedResultsCookie": string,
    "_sortKeys": "-timestamp"
});
```

For more information about `_queryFilter` syntax, refer to Common Filter Expressions. For more information about paging, refer to Page Query Results.

fields

list

A list of the fields that should be returned in the result. The list of fields can include wild cards, such as `*` or `*_ref`. The following example returns only the `userName` and `_id` fields:

```
openidm.query("managed/user", { "_queryFilter": "/userName sw \"user.1\""}, ["userName", "_id"]);
```

This parameter is particularly useful in enabling you to return the response from a query without including intermediary code to massage it into the right format.

Fields are specified as JSON pointers.

Returns

The result of the query. A query result includes the following parameters:

query-time-ms

(For JDBC repositories only) the time, in milliseconds, that IDM took to process the query.

result

The list of entries retrieved by the query. The result includes the properties that were requested in the query.

The following example shows the result of a custom query that requests the ID, user name, and email address of all managed users in the repository.

```
{
  "result": [
    {
      "_id": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
      "_rev": "00000000a059dc9f",
      "userName": "bjensen",
      "mail": "bjensen@example.com"
    },
    {
      "_id": "42f8a60e-2019-4110-a10d-7231c3578e2b",
      "_rev": "00000000d84ade1c",
      "userName": "scarter",
      "mail": "scarter@example.com"
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```


Throws

An exception is thrown if the given query could not be processed.

Examples

The following sample query uses a `_queryFilter` to query the managed user repository:

```
openidm.query("managed/user", {'_queryFilter': userIdPropertyName + ' eq "' + security.authenticationId  +
'"'});
```

The following sample query references the `for-userName` query, defined in the repository configuration, to query the managed user repository:

```
openidm.query("managed/user", {"_queryId": "for-userName", "uid": request.additionalParameters.uid });
```

This function performs an action on the specified resource object. The `resource` and `actionName` are required. All other parameters are optional.

Parameters

resource

string

The resource that the function acts upon, for example, `managed/user`.

actionName

string

The action to execute. Actions are used to represent functionality that is not covered by the standard methods for a resource (create, read, update, delete, patch, or query). In general, you should not use the `openidm.action` function for create, read, update, patch, delete or query operations. Instead, use the corresponding function specific to the operation (for example, `openidm.create`).

Using the operation-specific functions lets you benefit from the well-defined REST API, which follows the same pattern as all other standard resources in the system. Using the REST API enhances usability for your own API, and enforces the established patterns.

IDM-defined resources support a fixed set of actions. For user-defined resources (scriptable endpoints) you can implement whatever actions you require.

Supported Actions Per Resource

The following list outlines the supported actions for each resource or endpoint. The actions listed here are also supported over the REST interface.

Actions supported on the `authentication` endpoint (`authentication/*`)

reauthenticate

Actions supported on the configuration resource (`config/`)

No action parameter applies.

Actions supported on custom endpoints

Custom endpoints enable you to run arbitrary scripts through the REST URI, and are routed at `endpoint/name`, where name generally describes the purpose of the endpoint. For more information on custom endpoints, refer to Create custom endpoints to launch scripts. You can implement whatever actions you require on a custom endpoint. IDM uses custom endpoints in its workflow implementation. Those endpoints, and their actions are as follows:

`endpoint/getprocessforuser` - `create, complete`
`endpoint/gettasksview` - `create, complete`

Actions supported on the `external` endpoint

- `external/email` - `send`, for example:

```
{
    emailParams = {
        "from" : 'admin@example.com',
        "to" : user.mail,
        "subject" : 'Password expiry notification',
        "type" : 'text/plain',
        "body" : 'Your password will expire soon. Please change it!'
    }
    openidm.action("external/email", "send",  emailParams);
}
```

- `external/email` - `sendTemplate`, for example:

```
{
    emailParams = {
        "templateName" : "welcome",
        "to" : user.mail,
        "cc" : "ccUser1@example.com,ccUser2@example.com",
        "bcc" : "bigBoss@example.com"
    }
    openidm.action("external/email", "sendTemplate",  emailParams);
}
```

- `external/rest` - `call`, for example:

```
openidm.action("external/rest", "call", params);
```

Actions supported on the `info` endpoint (`info/*`)

No action parameter applies.

Actions supported on managed resources (`managed/*`)

patch, triggerSyncCheck

Actions supported on the policy resource (`policy`)

validateObject, validateProperty

For example:

```
openidm.action("policy/" + fullResourcePath, "validateObject", request.content, { "external" :
"true" });
```

Actions supported on the reconciliation resource (`recon`)

recon, reconById, cancel

For example:

```
openidm.action("recon/_id", "cancel", content, params);
```

> **Note**
>
> A *cancel* action requires the entire reconciliation resource path (_id).

Actions supported on the repository (`repo`)

command

For example:

```
var r, command = {
    "commandId": "purge-by-recon-number-of",
    "numberOf": numOfRecons,
    "includeMapping": includeMapping,
    "excludeMapping": excludeMapping
};
r = openidm.action("repo/audit/recon", "command", {}, command);
```

Actions supported on the script resource (`script`)

eval

For example:

```
openidm.action("script", "eval", getConfig(scriptConfig), {});
```

Actions supported on the synchronization resource (`sync`)

getLinkedResources, notifyCreate, notifyDelete, notifyUpdate, performAction

For example:

```
openidm.action('sync', 'performAction', content, params);
```

Actions supported on system resources (`system/*`)

availableConnectors, createCoreConfig, createFullConfig, test, testConfig, liveSync, authenticate, script

For example:

```
openidm.action("system/ldap/account", "authenticate", {"username" : "bjensen", "password" :
"Passw0rd"});
```

Actions supported on the task scanner resource (`taskscanner`)

execute, cancel

Actions supported on the workflow resource (`workflow/*`)

On `workflow/processdefinition` create, complete

On `workflow/processinstance` create, complete

For example:

```
var params = {
    "_key":"contractorOnboarding"
};
openidm.action('workflow/processinstance', 'create', params);
```

On `workflow/taskinstance` claim, create, complete

For example:

```
var params = {
    "userId":"manager1"
};
openidm.action('workflow/taskinstance/15', 'claim', params);
```

content

object

Content given to the action for processing.

params

object (optional)

Additional parameters passed to the script. The `params` object must be a set of simple key:value pairs, and cannot include complex values. The parameters must map directly to URL variables, which take the form `name1=val1&name2=val2&…`.

fields

JSON array (optional)

An array of the fields that should be returned in the result. The list of fields can include wild cards, such as `*` or `*_ref`. If no fields are specified, the entire object is returned.

Returns

The result of the action may be `null`.

Throws

If the action cannot be executed, an object containing an `error` property is returned.

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/CBC/PKCS5Padding`.

alias

string

The key alias in the keystore, such as `openidm-sym-default` (deprecated) or a purpose defined in the `secrets.json` file, such as `idm.password.encryption`.

Returns

The value, encrypted with the specified cipher and key.

Throws

An exception is thrown if the object could not be encrypted.

This function decrypts a value.

Parameters

value

object

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. An error is thrown if the value is passed in as a string - it must be passed in an object.

This function determines if a value is encrypted.

Parameters

object to check

any

The object whose value should be checked to determine if it is encrypted.

Returns

Boolean, `true` if the value is encrypted, and `false` if it is not encrypted.

Throws

An exception is thrown if the server is unable to detect whether the value is encrypted, for any reason.

This function calculates a value using a salted hash algorithm.

Supported Hashing Algorithms and Configuration Properties

Algorithm	Config Property and Description
`BCRYPT`	**`cost`** Value between 4 and 31. Default is `13`.
`PBKDF2`	**`hashLength`** Byte-length of the generated hash. Default is `16`. **`saltLength`** Byte-length of the salt value. Default is `16`. **`iterations`** Number of cryptographic iterations. Default is `20000`. **`hmac`** HMAC algorithm. Default is `SHA3-256`. Supported values: • `SHA-224` • `SHA-256` • `SHA-384` • `SHA-512` • `SHA3-224` • `SHA3-256` • `SHA3-384` • `SHA3-512`
`SCRYPT`	**`hashLength`** Byte-length of the generated hash, must be greater than or equal to 8. Default is `16`. **`saltLength`** Byte-length of the salt value. Default is `16`. **`n`** CPU/Memory cost. Must be greater than 1, a power of 2, and less than *2^(128 * r / 8)*. Default is `32768`. **`p`** Parallelization. Must be a positive integer less than or equal to *Integer.MAX_VALUE / (128 * r * 8)*. Default is `1`. **`r`** Block size. Must be greater than or equal to 1. Default is `8`.

Algorithm	Config Property and Description
`SHA-256`	**`saltLength`** Byte-length of the salt value. Default is `16`. **Note** This is the default hashing.
`SHA-384`	**`saltLength`** Byte-length of the salt value. Default is `16`.
`SHA-512`	**`saltLength`** Byte-length of the salt value. Default is `16`.

Parameters

value

any

The value to be hashed.

algorithm

string (optional)

The hashing algorithm. Example: `SHA-512`.

If no algorithm is provided, a `null` value must be passed, and the algorithm defaults to SHA-256.

options

For Groovy, Map or JSON value (optional)

For JavaScript, JSON object (optional)

Configuration properties for the selected algorithm.

Returns

The value, calculated with the specified hash algorithm.

Throws

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

Examples

Groovy (Map)

```
openidm.hash(\"dummy\", \"BCRYPT\", [\"cost\": 10]);
```

Groovy (JSON value)

```
JsonValue v = new JsonValue( [\"cost\": 10]); return openidm.hash(\"dummy\", \"BCRYPT\", v);
```

JavaScript

```
openidm.hash(\"dummy\", \"BCRYPT\", {\"cost\": 10})
```

This function detects whether a value has been calculated with a salted hash algorithm.

Parameters

value

any

The value to be reviewed.

Returns

Boolean, `true` if the value is hashed, and `false` otherwise.

Throws

An exception is thrown if the server is unable to detect whether the value is hashed, for any reason.

This function detects whether a string, when hashed, matches an existing hashed value.

Parameters

string

any

A string to be hashed.

value

any

A hashed value to compare to the string.

Returns

Boolean, `true` if the hash of the string matches the hashed value, and `false` otherwise.

Throws

An exception is thrown if the string could not be hashed.

> **Note**
>
> These functions can also have a vanity binding to make them more descriptive, such as `consent.action()` instead of `openidm.action("consent" ...)`, by setting `resourceBindings` when you declare the script. In this case, the syntax for these functions would omit the resource name from the function parameters.
> See Call a script from the IDM configuration for more information about `resourceBindings`.

Log functions

IDM also provides a `logger` object to access the Simple Logging Facade for Java (SLF4J) facilities. The following code shows an example of the `logger` object.

```
logger.info("Parameters passed in: {} {} {}", param1, param2, param3);
```

To set the log level for JavaScript scripts, add the following property to your project's `conf/logging.properties` file:

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

The level can be one of `SEVERE` (highest value), `WARNING, INFO, CONFIG, FINE, FINER`, or `FINEST` (lowest value). For example:

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

In addition, JavaScript has a useful logging function named `console.log()`. This function provides an easy way to dump data to the IDM standard output (usually the same output as the OSGi console). The function works well with the JavaScript built-in function `JSON.stringify` and provides fine-grained details about any given object. For example, the following line will print a formatted JSON structure that represents the HTTP request details to STDOUT.

```
console.log(JSON.stringify(context.http, null, 4));
```

> **Note**
>
> These logging functions apply only to JavaScript scripts. To use the logging functions in Groovy scripts, the following lines must be added to the Groovy scripts:
>
> ```
> import org.slf4j.*;
> logger = LoggerFactory.getLogger('logger');
> ```

The script engine supports the following log functions:

Logs a message at DEBUG level.

Parameters

message

string

The message format to log. Params replace `{}` in your message.

params

object

Arguments to include in the message.

Returns

A `null` value if successful.

Throws

An exception is thrown if the message could not be logged.

Logs a message at ERROR level.

Parameters

message

string

The message format to log. Params replace `{}` in your message.

params

object

Arguments to include in the message.

Returns

A `null` value if successful.

Throws

An exception is thrown if the message could not be logged.

Logs a message at INFO level.

Parameters

message

string

The message format to log. Params replace `{}` in your message.

params

object

Arguments to include in the message.

Returns

A `null` value if successful.

Throws

An exception is thrown if the message could not be logged.

Logs a message at TRACE level.

Parameters

message

string

The message format to log. Params replace `{}` in your message.

params

object

Arguments to include in the message.

Returns

A `null` value if successful.

Throws

An exception is thrown if the message could not be logged.

Logs a message at WARN level.

Parameters

message

string

The message format to log. Params replace `{}` in your message.

params

object

Arguments to include in the message.

Returns

A `null` value if successful.

Throws

An exception is thrown if the message could not be logged.

Workflow

Guide to enabling and using ForgeRock® Identity Management workflows.

IDM provides an embedded workflow and business process engine based on Flowable and the Business Process Model and Notation (BPMN) 2.0 standard. This guide describes how to configure the workflow engine, and how to manage workflow tasks and processes using the REST interface and the admin UI.

Workflow Tools

Learn about BPMN 2.0 and the workflow tools.

Enable Workflows

Configure the workflow engine and datasource to enable workflows.

Invoke Workflows

Learn where and how to trigger workflows.

Custom Workflows

Create custom workflow templates.

Note

Workflows are not supported with a DS repository. If you are using a DS repository for IDM data, you must configure a separate JDBC repository as the workflow datasource.

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

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

BPMN 2.0 and workflow 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.

BPMN 2.0 lets you add artifacts that describe your workflows and business processes to IDM, for provisioning and other purposes. You can create workflow definitions using a text editor or the ForgeRock Workflow Editor UI. The ForgeRock Workflow Editor UI is an open-source project that is not supported or included in IDM. This visual workflow editor can be set up as a standalone *node* server or integrated directly with IDM.

You can create scripts using Groovy and JavaScript.

Scripts inside BPMN 2.0 XML files have access to the following scripting variables:

- `openidm`
- `identityServer`
- `console`

For example, to log a message with Groovy:

```
console.log('my message')
```

Note

For more information about the graphical notations and XML representations for events, flows, gateways, tasks, process constructs, and more, refer to BPMN 2.0 Constructs.
IDM does not support the following constructs:

- Mule task
- Camel task

The following terms are reserved and cannot be used as variable names: `out`, `out:print`, `lang:import`, `context`, and `elcontext`.

Enable workflows

IDM embeds a Flowable Process Engine that starts in the OSGi container. Workflows are not active by default. IDM needs two configuration files to activate the workflow bundle:

`workflow.json`

The Flowable engine configuration, including the data source.

`datasource.jdbc-default.json`

The default data source for Flowable.

When you enable workflows in the admin UI, IDM creates `workflow.json` in your project's `conf/` directory.

1. Log in to the admin UI.
2. From the navigation bar, select **Configure > System Preferences**.
3. On the **System Preferences** page, click the **Workflow** tab.
4. Enable the display of workflows, and click **Save**.
5. Optionally, configure the workflow engine.
6. Configure the workflow data source.

Configure the workflow engine

IDM creates the default `workflow.json` file with the following structure:

```
{
    "useDataSource" : "default",
    "workflowDirectory" : "&{idm.instance.dir}/workflow",
    "userResource": {
        "path": "managed/user",
        "queryFilter": "/userName eq \"${username}\""
    },
    "groupResource": {
        "path": "managed/group",
        "queryFilter": "/id eq \"${gid}\""
    }
}
```

`useDataSource`

The datasource configuration file that points to the repository where Flowable should store data.

By default, this is the `datasource.jdbc-default.json` file. For information about changing the data store that Flowable uses, refer to Configure the Workflow Data Source.

`workflowDirectory`

Specifies the location where IDM expects to find workflow processes. By default, IDM looks for workflow processes in the `project-dir/workflow` directory.

In addition to these default properties, you can configure the Flowable engine `history` level:

```
{
    "history" : "audit"
}
```

When a workflow is executed, information can be logged as determined by the history level. The history level can be one of the following:

- `none` This level results in the best performance for workflow execution, but no historical information is retained.

- `activity` Logs all process instances and activity instances, without details.
- `audit` This is the default level. All process instances, activity instances, and submitted form properties are logged so that all user interaction through forms is traceable and can be audited.
- `full` This is the highest level of history logging and has the greatest performance impact. This history level stores all the information that is stored for the `audit` level, as well as any process variable updates.

Configure workflow email

Workflows can send an email using the following methods:

To use workflow email tasks, add the email configuration to `workflow.json`.

Example email configuration:

```
"mail" : {
    "host" : "mail.example.com",
    "port" : 1025,
    "username" : "username",
    "password" : "password",
    "useSSL" : false,
    "starttls" : true,
    "defaultFrom" : "workflow@example.com",
    "forceTo" : "overrideSendToEmail@example.com"
}
```

Only JavaScript and Groovy are supported as `ScriptTask#scriptFormat` languages.

Emails sent using `scriptTask` utilize IDM's email client configuration.

Example script:

```
openidm.action("external/email", "send", { "to": "bob@example.com" }, { waitForCompletion: true });
```

Configure the workflow data source

The Flowable engine requires a JDBC database. The connection details to the database are specified in the `datasource.jdbc-default.json` file. If you are using a JDBC repository for IDM data, you will already have a `datasource.jdbc-default.json` file in your project's `conf/` directory. In this case, when you enable workflows, IDM uses the existing JDBC repository and creates the required Flowable tables in that JDBC repository.

> **Important**
>
> If you are using a DS repository for IDM data, you must configure a separate JDBC repository as the workflow datasource. For more information, refer to Select a repository.

To specify a Flowable data source separate from your existing IDM repository, create a new *datasource* configuration file in your project's `conf/` directory (for example, `datasource.jdbc-flowable.json`) with the connection details to the separate data source. Then, reference that file in the `useDataSource` property of the `workflow.json` file (for example, `"useDataSource" : "flowable"`).

For more information about the fields in this file, refer to JDBC Connection Configuration.

Custom workflow object mapping

For custom object mapping, edit the default `workflow.json` configuration:

```
"userResource": {
    "path": "managed/user",
    "queryFilter": "/userName eq \"${username}\""
},
"groupResource": {
    "path": "managed/group",
    "queryFilter": "/id eq \"${gid}\""
}
```

Note

Do not replace `${username}` or `${gid}` in the queryFilter; for example:

- OK: `"queryFilter": "/callSign eq \"${username}\""`
- *NOT* OK: `"queryFilter": "/callSign eq \"${callsign}\""`

Test workflow integration

IDM reads workflow definitions from the `/path/to/openidm/workflow` directory.

The `/path/to/openidm/samples/provisioning-with-workflow/` sample provides a workflow definition (`contractorOnboarding.bar`) that you can use to test the workflow integration.

1. Create a `workflow` directory in your project directory and copy the sample workflow to that directory:

```
cd project-dir
mkdir workflow
cp samples/provisioning-with-workflow/workflow/contractorOnboarding.bar workflow/
```

2. Verify the workflow integration by using the REST API. The following REST call lists the defined workflows:

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

The result is similar to:

```
{
  "result": [
    {
      "_id": "contractorOnboarding:1:5",
      "_rev": "1",
      "candidateStarterGroupIdExpressions": [],
      "candidateStarterUserIdExpressions": [],
      "category": "Examples",
      "deploymentId": "1",
      "description": null,
      "eventSupport": {},
      "executionListeners": {},
      "graphicalNotationDefined": false,
      "hasStartFormKey": true,
      "historyLevel": null,
      "ioSpecification": null,
      "key": "contractorOnboarding",
      "laneSets": [],
      "name": "Contractor onboarding process",
      "participantProcess": null,
      "processDiagramResourceName": "contractorOnboarding.contractorOnboarding.png",
      "properties": {},
      "resourceName": "contractorOnboarding.bpmn20.xml",
      "revisionNext": 2,
      "startFormHandler": null,
      "suspended": false,
      "suspensionState": 1,
      "taskDefinitions": null,
      "tenantId": "",
      "variables": null,
      "version": 1
    }
  ],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```


For more information about the above workflow, refer to Provision users with workflow.

For more information about managing workflows over REST, refer to Workflows.

Create workflows

For more information about the graphical notations and XML representations for events, flows, gateways, tasks, process constructs, and more, refer to BPMN 2.0 Constructs.

IDM does not support the following constructs:

- Mule task
- Camel task

1. Create a workflow definition, and save it with a `bpmn20.xml` extension.

2. Package the workflow definition file in a Business Archive File (`.bar`). A `.bar` file is similar to a `.zip` file, but with a different extension.
3. Copy the `.bar` file to the `openidm/workflow` directory.
4. Invoke the workflow using a script (`openidm/script/`), or directly, using the REST interface. For more information, refer to Invoke workflows.

   You can also schedule the workflow to be invoked repeatedly, or at a future time.

Workflow definition comparison

Versions of IDM prior to 7.0 used the Activiti workflow engine. If you are upgrading from one of these versions, your current workflow definitions will continue to work in compatibility mode, but new definitions must be written for the new engine, Flowable. The following overview shows the main differences between the old and new workflow definitions:

> **Note**
>
> You can view additional upgrade information in the Flowable Migration Guide.

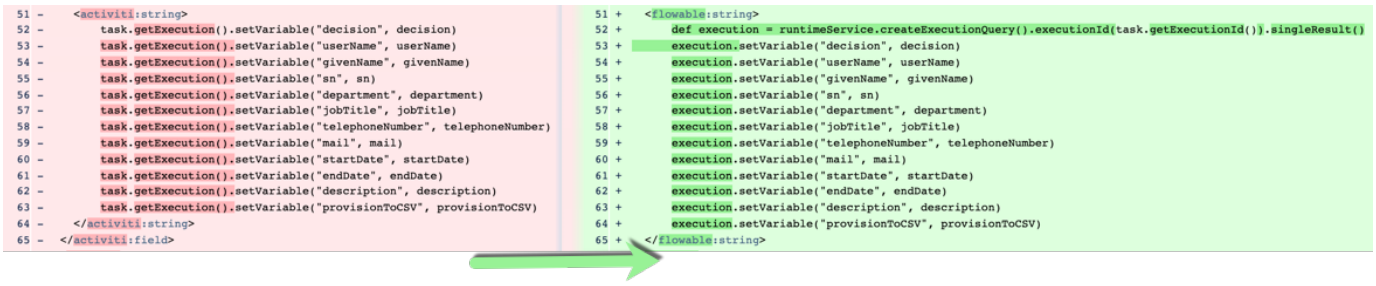
1. Change all occurrences of `activiti` to `flowable`. Update examples:
   - Namespace

     ```
     xmlns:flowable="http://flowable.org/bpmn"
     ```

   - Elements

     ```
     <flowable:formProperty id="givenName" name="First Name" type="string" required="true"></
     flowable:formProperty>
     <flowable:formProperty id="sn" name="Last Name" type="string" required="true"></flowable:formProperty>
     <flowable:formProperty id="department" name="Department" type="string"></flowable:formProperty>
     ```

2. Change `task.getExecution()` , per the changes to the sample file `contractorOnboarding.bpmn20.xml` :



Query workflows

The workflow implementation supports filtered queries that let you query the running process instances and tasks, based on specific query parameters. To perform a filtered query, send a GET request to the `workflow/processinstance` context path, including the query in the URL.

For example, the following query returns all process instances with the business key `"newOrder"`, as invoked in the previous example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/workflow/processinstance?_queryId=filtered-
query&processInstanceBusinessKey=newOrder"
```

Any workflow properties can be queried using the same notation; for example, `processDefinitionId=managedUserApproval:1:6405`. The query syntax applies to all queries with `_queryId=filtered-query`. The following query returns all process instances that were started by the user `openidm-admin`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/workflow/processinstance?_queryId=filtered-query&startUserId=openidm-admin"
```

You can also query process instances based on the value of any process instance variable, by prefixing the variable name with `var-`. For example:

```
var-processvariablename=processvariablevalue
```

Invoke workflows

You can invoke workflows and business processes from any trigger point within IDM, including reacting to situations discovered during reconciliation. Workflows can be invoked from script files, using the `openidm.create()` function, or directly from the REST interface.

The following sample script extract shows how to invoke a workflow from a script file:

```
/*
 * Calling 'myWorkflow' workflow
 */

var params = {
  "_key": "myWorkflow"
};

openidm.create('workflow/processinstance', null, params);
```

The `null` in this example indicates that you do not want to specify an ID as part of the create call. For more information, refer to `openidm.create()`.

You can invoke the same workflow from the REST interface with the following REST call:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{"_key":"myWorkflow"}' \
"http://localhost:8080/openidm/workflow/processinstance?_action=create"
```

For more information, refer to Workflows.

There are two ways in which you can specify the workflow definition that is used when a new workflow instance is started.

- `_key` specifies the `id` attribute of the workflow process definition, for example:

  ```
  <process id="sendNotificationProcess" name="Send Notification Process">
  ```

  If there is more than one workflow definition with the same `_key` parameter, the latest deployed version of the workflow definition is invoked.

- `_processDefinitionId` specifies the ID that is generated by the Flowable Process Engine when a workflow definition is deployed; for example:

  ```
  "sendNotificationProcess:1:104";
  ```

  To obtain the `processDefinitionId`, query the available workflows, for example:

```
{
  "result": [
    {
      "name": "Process Start Auto Generated Task Auto Generated",
      "_id": "ProcessSAGTAG:1:728"
    },
    {
      "name": "Process Start Auto Generated Task Empty",
      "_id": "ProcessSAGTE:1:725"
    },
    ...
  ]
}
```

If you specify a `_key` and a `_processDefinitionId`, the `_processDefinitionId` is used because it is more precise.

Use the optional `_businessKey` parameter to add specific business logic information to the workflow when it is invoked. For example, the following workflow invocation assigns the workflow a business key of `"newOrder"`. This business key can later be used to query "newOrder" processes.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{"_key":"myWorkflow", "_businessKey":"newOrder"}' \
"http://localhost:8080/openidm/workflow/processinstance?_action=create"
```

Access to workflows is based on IDM roles, and is configured in your project's `conf/process-access.json` file. For more information, refer to Secure Access to Workflows.

Workflow audit

The audit service logs workflow information in the activity event topic (default location: `openidm/audit/activity.audit.json`).

Example workflow audit events using the provisioning-with-workflow sample:

Each step shows the action performed along with the resulting audit data.

1. `user1` completes the *Contractor Onboarding Form*.

```
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-3871",
  "timestamp": "2020-05-06T17:39:52.021Z",
  "eventName": "workflow-create_process",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-3865",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "workflow/processinstance/6",
  "operation": "CREATE",
  "changedFields": [],
  "revision": null,
  "status": "SUCCESS",
  "message": "Process created. processDefinitionId = contractorOnboarding:1:5, processDefinitionKey = null,
businessKey = null",
  "passwordChanged": false
}
```


2. `manager1` self-assigns the task.


```
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5748",
  "timestamp": "2020-05-06T17:43:18.058Z",
  "eventName": "workflow-update_task",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5744",
  "userId": "manager1",
  "runAs": "manager1",
  "objectId": "workflow/taskinstance/36",
  "operation": "UPDATE",
  "changedFields": [
    "/assignee"
  ],
  "revision": null,
  "status": "SUCCESS",
  "message": "Task updated",
  "passwordChanged": false
}
```


> **Note**
>
> `"changedFields":["/assignee"]` only displays when `conf/audit.json` contains the property `"watchedFields" : [ "assignee" ]`. For a complete list of fields that can be watched in this situation, refer to the API Descriptor for `UPDATE workflow/taskinstance/`.

3. `manager1` completes the task. Notice that `transactionId` is correlated to all managed/user, and other, operations.

```
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5868",
  "timestamp": "2020-05-06T17:43:22.138Z",
  "eventName": "activity",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "000000001edd9dc2",
  "status": "SUCCESS",
  "message": "create",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5871",
  "timestamp": "2020-05-06T17:43:22.141Z",
  "eventName": "activity",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "internal/usermeta/cd237cca-913e-481e-9282-ba16c84b5131",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "0000000030b45c3e",
  "status": "SUCCESS",
  "message": "create",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5876",
  "timestamp": "2020-05-06T17:43:22.145Z",
  "eventName": "relationship_created",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2/authzRoles/ee5bbbce-a020-45db-
ab41-66c80d84d8be",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "00000000fe6da3a7",
  "status": "SUCCESS",
  "message": "Relationship originating from managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2 via the
relationship field authzRoles and referencing internal/role/openidm-authorized was created.",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5879",
  "timestamp": "2020-05-06T17:43:22.147Z",
  "eventName": "relationship_created",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2/manager/b58e5695-9e43-4e76-b89c-
e5d69d3bf52d",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "000000008dcca1b6",
  "status": "SUCCESS",
```

```
  "message": "Relationship originating from managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2 via the
relationship field manager and referencing managed/user/038e65de-95ce-4180-94d3-4ea64bf25c6b was created.",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5882",
  "timestamp": "2020-05-06T17:43:22.149Z",
  "eventName": "relationship_created",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2/_meta/d9299603-b768-44b6-a4c9-9b6441ca212e",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "0000000027b29fb4",
  "status": "SUCCESS",
  "message": "Relationship originating from managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2 via the
relationship field _meta and referencing internal/usermeta/cd237cca-913e-481e-9282-ba16c84b5131 was created.",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5908",
  "timestamp": "2020-05-06T17:43:22.778Z",
  "eventName": "activity",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "internal/notification/12aa1698-bb1e-42c6-a92d-2e959c217ad0",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "000000004d025d75",
  "status": "SUCCESS",
  "message": "create",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5911",
  "timestamp": "2020-05-06T17:43:22.781Z",
  "eventName": "relationship_created",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "internal/notification/12aa1698-bb1e-42c6-a92d-2e959c217ad0/target/eec80d30-be1e-4c5d-9873-
b4395373c833",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "00000000b4c7a701",
  "status": "SUCCESS",
  "message": "Relationship originating from internal/notification/12aa1698-bb1e-42c6-a92d-2e959c217ad0 via the
relationship field target and referencing managed/user/038e65de-95ce-4180-94d3-4ea64bf25c6b was created.",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5920",
  "timestamp": "2020-05-06T17:43:22.791Z",
  "eventName": "activity",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "internal/notification/eec030e5-e520-4cf1-99c2-a9bbecb0627b",
  "operation": "CREATE",
  "changedFields": [],
```

```
  "revision": "0000000033465ada",
  "status": "SUCCESS",
  "message": "create",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5923",
  "timestamp": "2020-05-06T17:43:22.794Z",
  "eventName": "relationship_created",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "internal/notification/eec030e5-e520-4cf1-99c2-a9bbecb0627b/target/11eb13f8-991f-45ea-95dc-
e8f0fd0b95c3",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "000000005134a80d",
  "status": "SUCCESS",
  "message": "Relationship originating from internal/notification/eec030e5-e520-4cf1-99c2-a9bbecb0627b via the
relationship field target and referencing managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2 was created.",
  "passwordChanged": false
}
```


4. The audit service logs the `workflow-complete_task` event.


```
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5926",
  "timestamp": "2020-05-06T17:43:22.827Z",
  "eventName": "workflow-complete_task",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "manager1",
  "runAs": "manager1",
  "objectId": "workflow/taskinstance/36",
  "operation": "complete",
  "changedFields": [],
  "revision": null,
  "status": "SUCCESS",
  "message": "Task completed",
  "passwordChanged": false
}
```


Custom workflow templates

The embedded workflow engine integrates with the default End User UI. For simple custom workflows, you can use the standard Flowable form properties, and have the UI render the corresponding generic forms automatically. For more complex functionality, including input validation, rich input field types, complex CSS, and more, you must define a custom form template.

The default workflows provided with IDM use the Vue JS framework for display in the End User UI. To write a custom form template, you must have a basic understanding of the Vue JS framework and how to create components. A sample workflow template is provided at `/path/to/samples/provisioning-with-workflow/workflow/contractorOnboarding.bar`. To extract the archive, run the following command:

```
jar -xvf contractorOnboarding.bar
inflated: contractorForm.js
inflated: contractorOnboarding.bpmn20.xml
```

The archive includes the workflow definition `contactorOnboarding.bpmn20.xml` and the corresponding JavaScript template `contractorForm.js` to render the workflow in the UI.

Password synchronization plugins

> *Guide to configuring and integrating the password synchronization plugins into your IDM deployment.*

Password synchronization ensures uniform password changes across the resources that store the password. This guide shows you how to use password synchronization plugins to synchronize passwords between ForgeRock Identity Management (IDM) and an LDAP server, either ForgeRock Directory Services (DS) or Active Directory.

Password Sync Plugins

Discover how password synchronization works in IDM and learn about the plugins provided for synchronizing passwords with LDAP servers.

Synchronize With DS

Use the ForgeRock Directory Services password synchronization plugin.

Synchronize With AD

Use the Active Directory password synchronization plugin.

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

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

Password synchronization plugins

Password synchronization ensures uniform password changes across the resources that store the password. After password synchronization, a user can authenticate with 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.

IDM can propagate passwords to the resources that store a user's password. In addition, you can download plugins from the Backstage download site to intercept and synchronize passwords that are changed natively in ForgeRock Directory Services (DS) and Active Directory.

If you use these plugins to synchronize passwords, set up password policy enforcement on the LDAP resource, rather than on IDM. Alternatively, ensure that all password policies that are enforced are identical to prevent password updates on one resource from being rejected by IDM or by another resource.

The password synchronization plugin intercepts password changes on the LDAP resource before the passwords are stored in encrypted form. The plugin then sends the intercepted password value to IDM, using an HTTP POST request to patch the corresponding managed user object.

> **Note**
>
> The plugins do not use the LDAP connector to transmit passwords, but send a generic HTTP POST request with a `patch` action.

If the IDM instance is unavailable when a password is changed in either DS or Active Directory, the respective password plugin intercepts the change, encrypts the password, and stores the encrypted password in a JSON file. The plugin then checks whether the IDM instance is available, at a predefined interval. When IDM becomes available, the plugin performs a PATCH on the managed user record, to replace the password with the encrypted password stored in the JSON file.

To be able to synchronize passwords, both password synchronization plugins require that the corresponding managed user object exist in the IDM repository.

Synchronize passwords with DS

The ForgeRock Directory Services (DS) password synchronization plugin intercepts passwords that are changed natively in the DS server and propagates these password changes to IDM. The password synchronization plugin captures password changes in clear text, encrypts them, and transmits them to IDM. If IDM is unavailable when a password change occurs, the password change is queued for subsequent retry.

The password synchronization plugin requires keys to encrypt changed passwords and certificates to secure communication between DS and IDM. The examples that follow use the keys generated when you set up the DS and IDM servers.

Set up IDM and DS

The following examples prepare a demonstration of password synchronization from DS to IDM. After this preparation:

- DS and IDM are installed and running on your computer, with default security settings.

In particular, both servers have key pairs used later in the demonstration:

- DS has a generated TLS key pair (alias: `ssl-key-pair` and certificate subject DN: `CN=DS, O=ForgeRock.com`) signed by the DS deploymentId-based CA (certificate subject DN: `CN=Deployment key, O=ForgeRock.com`).

  DS uses the certificate to set up TLS connections, and to authenticate to IDM.

- IDM has a generated key pair, alias `openidm-localhost`. The certificate is self-signed, and has certificate subject DN `CN=openidm-localhost, O=OpenIDM Self-Signed Certificate, OU=None, L=None, ST=None, C=None`.

  DS uses the public key certificate to encrypt passwords before sending them to IDM.

- IDM does not otherwise synchronize DS data with its own.

  This lets you confirm that DS, not synchronization, provides the updated password to IDM.

- DS and IDM both have a user account for Barbara Jensen.

  After you have configured password synchronization, when Barbara Jensen's password changes in DS, DS sends the change to IDM.

Prepare IDM

1. Update your hosts file.

   The IDM self-signed certificate uses the domain alias `openidm-localhost`. When testing the DS plugin on your computer, add the alias to your `/etc/hosts` file:

   ```
   127.0.0.1       localhost openidm-localhost
   ```

2. Unzip IDM.

3. Start IDM:

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

4. Add Barbara Jensen's account to IDM:

```
curl \
--request PUT \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept: application/json" \
--header "If-None-Match: *" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--data
'{"userName":"bjensen","password":"Password1","mail":"bjensen@example.com","sn":"Jensen","givenName":"Barbara"
}' \
"http://localhost:8080/openidm/managed/user/bjensen"
{
  "_id": "bjensen",
  "_rev": "revision",
  "userName": "bjensen",
  "mail": "bjensen@example.com",
  "sn": "Jensen",
  "givenName": "Barbara",
  "accountStatus": "active",
  "effectiveAssignments": [],
  "effectiveRoles": []
}
```

Prepare DS

1. Download DS 7.3 .zip distribution.
2. Unzip DS:

```
unzip -q ~/Downloads/DS-7.3.zip -d /path/to
```

3. Generate a DS deploymentId for DS setup and managing deployments:

```
/path/to/opendj/bin/dskeymgr create-deployment-id --deploymentIdPassword password
your-deployment-ID
```

4. Set up and start DS with data from an IDM example that includes Barbara Jensen's entry:

```
/path/to/opendj/setup \
--serverId evaluation-only \
--deploymentId your-deployment-ID \
--deploymentIdPassword password \
--rootUserDn uid=admin \
--rootUserPassword password \
--hostname localhost \
--adminConnectorPort 4444 \
--ldapsPort 1636 \
--profile ds-user-data \
--set ds-user-data/baseDn:dc=com \
--set ds-user-data/ldifFile:/path/to/openidm/samples/sync-with-ldap/data/Example.ldif \
--start \
--acceptLicense
```

The sample data includes an entry for Barbara Jensen. The rest of the sample configuration is not used here.

5. Check that the directory superuser can read Barbara Jensen's entry in the directory:

```
/path/to/opendj/bin/ldapsearch \
--port 1636 \
--useSSL \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--hostname localhost \
--bindDn uid=admin \
--bindPassword password \
--baseDn dc=com \
"(uid=bjensen)"
dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
cn: Barbara Jensen
description: Created for OpenIDM
givenName: Barbara
mail: bjensen@example.com
sn: Jensen
telephoneNumber: 1-360-229-7105
uid: bjensen
userPassword: {PBKDF2-HMAC-SHA256}10:hash
```

Notice that this entry differs from the account you added to IDM. However, the user identifier is `bjensen` in both cases. This will let IDM identify Barbara Jensen's account as the one whose password has changed when it receives the notification from DS.

Secure communication between IDM and DS

The password synchronization plugin encrypts passwords using IDM's public key. IDM then uses its private key to decrypt the password.

This section describes how to export IDM's certificate, containing its public key, to DS so that the password synchronization plugin can use the public key to encrypt the password. The same certificate is used by the plugin to trust the SSL certificate that is provided by IDM.

There are four possible modes of communication between the DS password synchronization plugin and IDM:

SSL Authentication

For this communication mode, you must import IDM's certificate into the DS truststore (either the self-signed certificate that is generated the first time IDM starts, or a CA-signed certificate).

Mutual SSL Authentication

> **Note**
>
> Mutual SSL authentication is the default configuration of the password synchronization plugin.

For this communication mode, you must:

- Import the IDM certificate into the DS truststore.
- Import the DS CA certificate into the IDM truststore.
- Add the DS certificate subject DN as a value of the `allowedAuthenticationIdPatterns` property in your project's `conf/authentication.json` file.

AM Bearer Tokens

When you use IDM and AM together as a platform, configure the password synchronization plugin to use AM bearer tokens for authentication.

For this communication mode, you must:

- Import the IDM certificate into the DS truststore.
- Import the DS CA certificate into the IDM truststore.
- Configure the password synchronization plugin to accept AM bearer tokens.

HTTP Basic Authentication

> **Warning**
>
> IDM supports basic authentication for ***testing purposes only***. Do *not* use basic authentication in production.

For this mode, the connection is secured using a username and password, rather than any exchange of certificates. Because the password sync plugin requires the IDM certificate to encrypt/decrypt passwords, you must import the IDM certificate into the DS truststore.

For this communication mode, you must:

- Import the IDM certificate into the DS truststore.
- Set the following properties in the plugin configuration:
  - `openidm-url`

- `openidm-username`
- `openidm-password`

Enable DS to trust the IDM certificate

The first time IDM starts, it generates a self-signed certificate. This procedure uses the self-signed certificate to demonstrate how to get the password synchronization plugin up and running. In a production environment, use a certificate that has been signed by a Certificate Authority (CA).

The default Java truststore contains signing certificates from well-known CAs. If your CA certificate is not in the default truststore, or if you are using a self-signed certificate, import it into the DS keystore, as described here.

1. Export the IDM self-signed certificate to a file, as follows:

```
keytool \
-export \
-alias openidm-localhost \
-file openidm-localhost.crt \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype jceks \
-storepass changeit
Certificate stored in file <openidm-localhost.crt>
```

   The default IDM keystore password is `changeit`.

2. Import the self-signed certificate into the DS keystore:

```
keytool \
-import \
-alias openidm-localhost \
-file openidm-localhost.crt \
-keystore /path/to/opendj/config/keystore \
-storepass:file /path/to/opendj/config/keystore.pin \
-storetype PKCS12 \
-noprompt
Certificate was added to keystore
```

3. Check that the IDM certificate is in the DS keystore:

```
keytool \
-list \
-keystore /path/to/opendj/config/keystore \
-storepass:file /path/to/opendj/config/keystore.pin
...
openidm-localhost, date, trustedCertEntry,
Certificate fingerprint (SHA-256): fingerprint
...
```

Enable IDM to trust DS certificates

For mutual SSL authentication, you must also import a trusted DS certificate into the IDM truststore, either a trusted CA certificate, or the CA certificate that is generated by the DS deploymentId and deploymentIdPassword. For more information, refer to Deployment IDs in the *DS Security Guide*. This procedure uses the CA certificate generated by the DS deploymentId and deploymentIdPassword.

1. Run the following command on your DS server to export the CA certificate to a file. Substitute the values for `--deploymentId` and `--deploymentIdPassword` with the values from when you set up the DS server:

```
/path/to/opendj/bin/dskeymgr \
export-ca-cert \
--deploymentId your-deployment-ID \
--deploymentIdPassword password \
--outputFile ssl-key-pair.pem
```

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

```
keytool \
-importcert \
-alias ssl-key-pair \
-keystore /path/to/openidm/security/truststore \
-storepass changeit \
-file ssl-key-pair.pem
Owner: CN=Deployment key, O=ForgeRock.com
Issuer: CN=Deployment key, O=ForgeRock.com
...
Trust this certificate? [no]: yes
Certificate was added to keystore
```

3. Check that the DS CA certificate is in the IDM truststore:

```
keytool \
-list \
-keystore /path/to/openidm/security/truststore \
-storepass changeit...
ssl-key-pair, date, trustedCertEntry,
Certificate fingerprint (SHA-256): fingerprint...
```

4. Restart IDM:

```
/path/to/openidm/shutdown.sh; /path/to/openidm/startup.sh
```

Configure the plugin for AM bearer tokens

This procedure uses the Platform Setup Guide as the basis for setting up IDM to use AM bearer tokens for authentication, and may need adjustment for your specific environment.

1. Perform Platform Setup - Deployment Two - Shared Identity Store, using the following settings during the applicable steps:
    - adminConnectorPort 4444
    - ldapPort 1389
    - enableStartTls
    - ldapsPort 1636
    - httpsPort 8443
    - replicationPort 8989
    - deploymentId your-deployment-ID
    - adminConnectorPort 4445
    - ldapPort 1390
    - ldapsPort 1637
    - replicationPort 8990
    - deploymentId your-deployment-ID
    - port: 8080
    - redirects: 8444
    - openidm.port.http=8081
    - openidm.port.https=8445
    - openidm.port.mutualauth=8446
    - openidm.host=openidm.example.com
    - openidm.auth.clientauthonlyports=8446
2. Configure a `ds-password-sync-plugin` OAuth Client for the Password Sync Plugin:
    - If you're not currently logged in to the AM console as the `amAdmin` user, log in.
    - In the Top Level Realm, select **Applications > OAuth 2.0 > Clients** , and click **Add Client** .
    - Enter the following details:
        - **Client ID** : `ds-password-sync-plugin`
        - **Client secret** : `ds-password-sync-plugin`
        - **Scopes** : `fr:idm:*, openid, am-introspect-all-tokens, am-introspect-all-tokens-any-realm`

- Click **Create** .
- On the **Advanced** tab:
  - **Token Endpoint Authentication Method** : Select `client_secret_basic` .
  - **Grant Types** : Add `Client Credentials` .
- Click **Save Changes** .

3. If you haven't performed the following procedures, do that now:
   - Import the IDM certificate into the DS truststore.
   - Import the DS CA certificate into the IDM truststore.
4. Add `openidm-localhost` to the `idm.default` mapping alias array in your project's `conf/secrets.json` file:

```
"mappings": [
  {
    "secretId": "idm.default",
    "types": [ "ENCRYPT", "DECRYPT" ],
    "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}", "openidm-localhost" ]
  },
  ...
]
```

5. Log in to the IDM admin UI as `amAdmin` and create a new managed/user:
   - From the navigation bar of the admin UI, select **Manage > User** , and then click **+ New User** .
   - On the **New User** page, enter the **Username** `ds-password-sync-plugin` , other information, as necessary, and click **Save** .
   - On the **ds-password-sync-plugin** user page, select the **Authorization Roles** tab.
   - Click **Add Authorization Roles** , select all of the following roles, and then click **Add** :

     `openidm-admin`
     `openidm-authorized`
     `openidm-cert`
     `openidm-reg`
6. Add a `staticUserMapping` for the `ds-password-sync-plugin` user to the `conf/authentication.json` file:

```
{
  "subject" : "ds-password-sync-plugin",
  "localUser" : "managed/user/ds-password-sync-plugin",
  "roles" : [
    "internal/role/openidm-authorized",
    "internal/role/openidm-admin"
  ]
}
```

Install and configure the DS plugin

The following steps install the password synchronization plugin on a DS directory server that is running on the same host as IDM (localhost). If you are running DS on a different host, use the fully qualified domain name instead of `localhost`.

You must use the plugin version that corresponds to your IDM and DS versions. For more information, refer to Supported Password Synchronization Plugins. This procedure assumes that you are using IDM 7.3, DS 7.3, and version 7.3 of the password synchronization plugin.

Depending on whether you are using IDM with AM, select one of the following plugin installation and configuration procedures:

For regular IDM authentication

1. Download the password synchronization plugin.
2. Extract the .zip file contents to the DS installation directory:

```
unzip ~/Downloads/DS-IDM-account-change-notification-handler-7.3.zip -d /path/to/opendj/
```

3. Restart DS to load the additional schema from the password synchronization plugin:

```
/path/to/opendj/bin/stop-ds --restart
Stopping Server...
...msg=Loaded extension from file '/path/to/opendj/lib/extensions/opendj-openidm-account-change-
notification-handler-7.3.jar'
...
...msg=The Directory Server has started successfully
```

4. Configure the password synchronization plugin:

```
/path/to/opendj/bin/dsconfig \
create-account-status-notification-handler \
--type openidm \
--handler-name "OpenIDM Notification Handler" \
--set enabled:true \
--set openidm-url:https://openidm-localhost:8444/openidm/managed/user \
--set private-key-alias:openidm-localhost \
--set certificate-subject-dn:"CN=openidm-localhost, O=OpenIDM Self-Signed Certificate, OU=None,
L=None, ST=None, C=None" \
--set ssl-cert-nickname:ssl-key-pair \
--set key-manager-provider:PKCS12 \
--set trust-manager-provider:PKCS12 \
--set password-attribute:password \
--set attribute-type:entryUUID \
--set attribute-type:uid \
--set query-id:for-userName \
--set log-file:logs/pwsync \
--set update-interval:5s \
--set request-retry-attempts:5000 \
--hostname localhost \
--port 4444 \
--bindDn uid=admin \
--bindPassword password \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--no-prompt
The Openidm Account Status Notification Handler was created successfully
```

Adapt the settings to match your DS and IDM deployments:

Setting	Details
`enabled`	Enables the plugin. Leave this setting as shown.
`openidm-url`	The endpoint where the plugin finds IDM managed user accounts. Port `8444` is the IDM default for mutual TLS connections.
`private-key-alias`	The IDM private key, used to decrypt the JSON objects from DS that contain passwords. The example references the default IDM private key of the self-signed key pair generated at setup time.
`certificate-subject-dn`	The certificate subject DN for the IDM public key. The DS plugin encrypts JSON objects with the IDM public key, so this must match the certificate for the IDM private key specified for the `private-key-alias` property. The example shows the subject DN of the default IDM self-signed certificate.

Setting	Details
`ssl-cert-nickname`	The alias of the DS TLS certificate used to authenticate to IDM. The example uses the default DS server certificate generated at setup time.
`key-manager-provider`	The provider for the keystore where DS finds its TLS key pair specified in `ssl-cert-nickname`. The example uses the default DS provider configured at setup time.
`trust-manager-provider`	The provider for the keystore where DS finds the IDM public key specified in `certificate-subject-dn`. The example uses the default DS provider configured at setup time, and updated in Enable DS to Trust the IDM Certificate.
`password-attribute`	The name of the password field in the JSON that DS sends to IDM for a password change. This attribute type must be defined in the managed object schema in IDM, and it must have either the user password or auth password syntax.
`attribute-type`	LDAP attributes that the DS plugin sends to IDM with the password. IDM can use these to uniquely identify the user, even if the user's account has moved. If no attribute types are specified, DS sends only the DN and the new password to IDM.
`query-id`	The query-id for the patch-by-query request. Leave this setting as shown.
`log-file`	The DS directory where the plugin writes log files containing encrypted passwords before notifying IDM. This setting has no effect in the example, where the `update-interval` is zero seconds.
`update-interval`	The interval at which the DS plugin sends password changes to IDM. If this value is zero, the plugin sends updates synchronously. No encrypted passwords are stored in the configured `log-file` directory. The plugin does not retry failed requests, irrespective of the `request-retry-attempts` setting.

Setting	Details
`request-retry-attempts`	The number of times the plugin attempts a synchronization request if the first attempt fails. If this value is zero, the request is not retried. If the value is greater than zero, the plugin retries the specified number of times before giving up and removing the request from its queue. When a request fails due to a transient condition, such as failure to contact IDM, or a connection timeout, the plugin does not decrement the number of retry attempts. The plugin logs a message with the reason the request failed, and continues to retry until IDM responds.

5. Restart DS for the new configuration to take effect:

```
/path/to/opendj/bin/stop-ds --restart
```

6. Update DS password policies to use the password synchronization plugin.

   The following example updates the default DS password policy:

```
/path/to/opendj/bin/dsconfig \
set-password-policy-prop \
--policy-name "Default Password Policy" \
--set account-status-notification-handler:"OpenIDM Notification Handler" \
--hostname localhost \
--port 4444 \
--bindDN uid=admin \
--bindPassword password \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--no-prompt
```

   For details on configuring DS password policies, refer to Passwords in the *DS Security Guide*.

Configure IDM for password synchronization

The password synchronization plugin uses client certificate authentication to authenticate to IDM. You must also update your security configuration to add the IDM key alias.

1. If your authentication configuration does not already include client certificate authentication, configure it as follows:
   1. Add the `CLIENT_CERT` authentication module to your authentication configuration.
   2. Set the `allowedAuthenticationIdPatterns` property to the certificate DN of the DS SSL certificate (`ssl-key-pair` by default).
   3. Add `internal/role/openidm-cert` to the array of `defaultUserRoles` .

The following example assumes that you are using the default DS `ssl-key-pair` certificate that has a certificate subject DN of `CN=DS, O=ForgeRock`:

```
"authModules" : [
    ...
    {
        "name" : "CLIENT_CERT",
        "properties" : {
            "queryOnResource" : "managed/user",
            "defaultUserRoles" : [
                "internal/role/openidm-cert",
                "internal/role/openidm-authorized"
            ],
            "allowedAuthenticationIdPatterns" : [
                ".*CN=DS, O=ForgeRock.com.*"
            ]
        },
        "enabled" : true
    },
    ...
]
```

For more information about client certificate authentication, refer to CLIENT_CERT.

2. Update the IDM secret store (`conf/secrets.json`) to add the alias used in the `private-key-alias` plugin setting to the `idm.default` secretId:

```
"mappings": [
    {
        "secretId" : "idm.default",
        "types": [ "ENCRYPT", "DECRYPT" ],
        "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}", "openidm-localhost" ]
    },
    ...
]
```

For more information about secret stores, refer to Secret stores.

For authentication with AM bearer tokens

1. Download the password synchronization plugin.
2. Extract the .zip file contents to the DS identity store installation directory:

```
unzip ~/Downloads/DS-IDM-account-change-notification-handler-7.3.zip -d /path/to/opendj-identity/
```

3. Configure the password synchronization plugin to use AM bearer token authentication:

```
/path/to/opendj-identity/bin/dsconfig \
create-account-status-notification-handler \
--type openidm \
--handler-name "OpenIDM Notification Handler" \
--set certificate-subject-dn:"CN=openidm-localhost,O=OpenIDM Self-Signed
Certificate,OU=None,L=None,ST=None,C=None" \
--set enabled:true \
--set attribute-type:entryUUID \
--set attribute-type:uid \
--set trust-manager-provider:PKCS12 \
--set key-manager-provider:PKCS12 \
--hostname identities.example.com \
--port 4445 \
--bindDn uid=admin \
--trustAll \
--bindPassword str0ngAdm1nPa55word \
--set log-file:logs/pwsync \
--set password-attribute:password \
--set query-id:for-userName \
--set private-key-alias:openidm-localhost \
--set openidm-url:https://openidm-localhost:8445/openidm/managed/user \
--set oauth2-access-token-url:http://am.example.com:8080/openam/oauth2/realms/root/access_token \
--set oauth2-scope:"openid fr:idm:*" \
--set oauth2-client-id:ds-password-sync-plugin \
--set oauth2-client-secret:ds-password-sync-plugin \
--set request-retry-attempts:5000 \
--set update-interval:5s \
--no-prompt
The Openidm Account Status Notification Handler was created successfully
```

Note

Adapt the above settings to match your DS, IDM, and AM deployments.

4. Restart DS for the new configuration to take effect:

```
/path/to/opendj-identity/bin/stop-ds --restart
```

5. Configure the DS password policy to use the password synchronization plugin. The following example updates the default DS password policy:

```
/path/to/opendj-identity/bin/dsconfig \
set-password-policy-prop \
--policy-name "Default Password Policy" \
--set account-status-notification-handler:"OpenIDM Notification Handler" \
--hostname identities.example.com \
--port 4445 \
--bindDN uid=admin \
--bindPassword str0ngAdm1nPa55word \
--usePkcs12TrustStore /path/to/opendj-identity/config/keystore \
--trustStorePassword:file /path/to/opendj-identity/config/keystore.pin \
--no-prompt
```

For details on configuring DS password policies, refer to Passwords in the *DS Security Guide* .

6. Generate an AM bearer token. For example:

```
curl -k \
--request POST \
--user "ds-password-sync-plugin:ds-password-sync-plugin" \
--data "grant_type=client_credentials" \
--data "scope=openid fr:idm:*" \
"http://am.example.com:8080/openam/oauth2/realms/root/access_token"
{
  "access_token": "access_token",
  "scope": "openid fr:idm:*",
  "id_token": "id_token",
  "token_type": "Bearer",
  "expires_in": 3599
}
```

7. Optionally, to test the AM bearer token, create a new managed user using the token as the authorization. For example:

```
curl -v \
--header "Authorization: Bearer access_token" \
--header "accept: application/json" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST "http://openidm-localhost:8081/openidm/managed/user?_action=create" \
--data '{"userName":"jdoe", "password": "6fNcHgBF", "mail": "jdoe@example.com", "sn": "Doe",
"givenName": "Jane"}'
{
  "_id": "7884b8ab-a226-4ee5-b9b9-0718f5a19335",
  "_rev": "00000000f527116e",
  "userName": "jdoe",
  "accountStatus": "active",
  "givenName": "Jane",
  "sn": "Doe",
  "mail": "jdoe@example.com"
}
```

Test the DS plugin

With the plugin installed and configured, and with secure communications enabled between DS and IDM, you can test that the setup has been successful as follows:

1. Change a user password in DS:

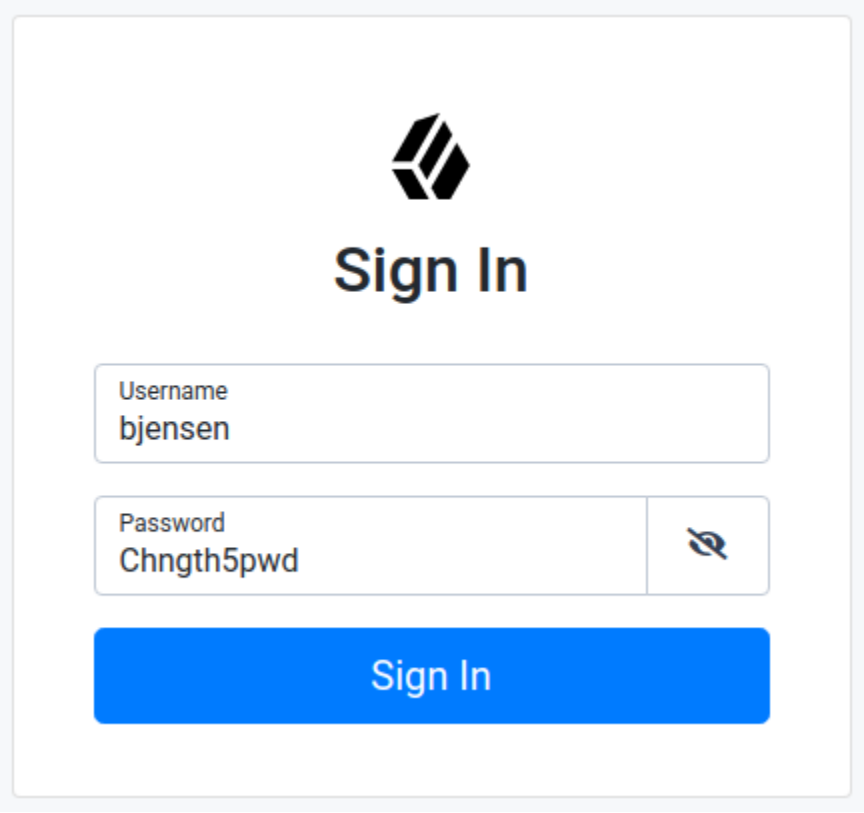
```
/path/to/opendj/bin/ldappasswordmodify \
--hostname localhost \
--port 1636 \
--useSsl \
--usePkcs12TrustStore /path/to/opendj/config/keystore \
--trustStorePassword:file /path/to/opendj/config/keystore.pin \
--bindDN uid=admin \
--bindPassword password \
--authzID dn:uid=bjensen,ou=people,dc=example,dc=com \
--newPassword Chngth5pwd
The LDAP password modify operation was successful
```

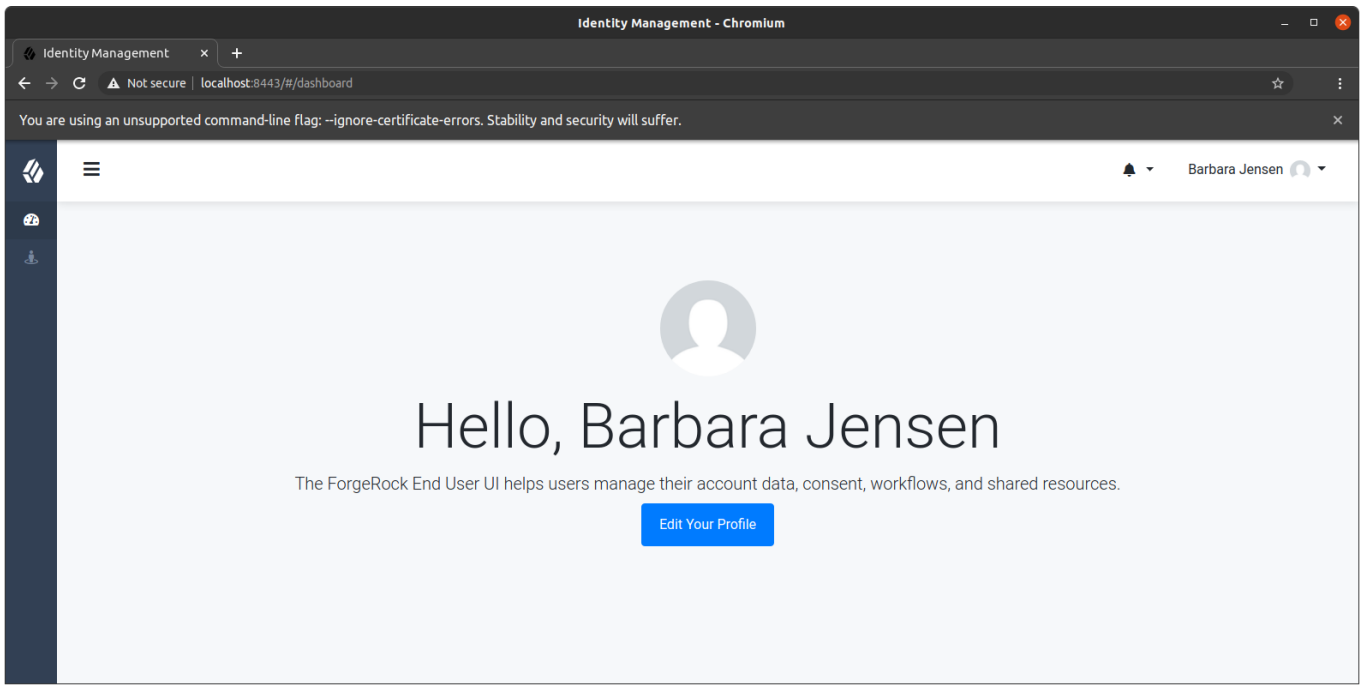
   The message `The LDAP password modify operation was successful` only indicates that the password change succeeded for DS. This does not mean that DS has propagated the change to IDM.

   When you have successfully updated the password in DS, DS attempts to synchronize the change to the corresponding IDM managed user account.

2. You should now be able to log in to the Self Service UI (https://localhost:8443/#login/) as that user ID with the new password.

```
chromium --ignore-certificate-errors https://localhost:8443/#login
```





Update the DS plugin

Additional steps may be necessary when updating the DS password synchronization plugin after upgrading DS. Check the corresponding Knowledge Base article for more information.

Uninstall the DS plugin

To uninstall the plugin, change the DS configuration as follows:

1. Reset your DS password policy configuration so that it no longer uses the password synchronization plugin.

   The following command resets the default password policy:

   ```
   /path/to/opendj/bin/dsconfig \
   set-password-policy-prop \
   --policy-name "Default Password Policy" \
   --reset account-status-notification-handler \
   --hostname localhost \
   --port 4444 \
   --bindDN uid=admin \
   --bindPassword password \
   --usePkcs12TrustStore /path/to/opendj/config/keystore \
   --trustStorePassword:file /path/to/opendj/config/keystore.pin \
   --no-prompt
   ```

2. Delete the IDM Notification Handler from the DS configuration:

   ```
   /path/to/opendj/bin/dsconfig \
   delete-account-status-notification-handler \
   --handler-name "OpenIDM Notification Handler" \
   --hostname localhost \
   --port 4444 \
   --bindDN uid=admin \
   --bindPassword password \
   --usePkcs12TrustStore /path/to/opendj/config/keystore \
   --trustStorePassword:file /path/to/opendj/config/keystore.pin \
   --no-prompt
   The Account Status Notification Handler was deleted successfully
   ```

3. Remove the password synchronization plugin from the DS extensions:

   ```
   rm /path/to/opendj/lib/extensions/opendj-openidm-account-change-notification-handler
   ```

4. Restart DS for the new configuration to take effect:

   ```
   /path/to/opendj/bin/stop-ds --restart
   ```

Synchronize passwords with Active Directory

Use the Active Directory (AD) password synchronization plugin to synchronize passwords between IDM and Active Directory (on systems running at least Microsoft Windows Server 2012 R2).

Install the plugin on Active Directory domain controllers (DCs) to intercept password changes, and send the password values to IDM over an encrypted channel. You must have Administrator privileges to install the plugin. In a clustered Active Directory environment, you must install the plugin on all DCs.

Install the Active Directory password synchronization plugin

The following steps install the password synchronization plugin on an Active Directory server:

1. Download the Active Directory password synchronization plugin.
2. Launch the installation wizard using one of the following methods:
   - In Windows Explorer, double-click the **ad-passwordchange-handler-1.7.0.exe** file.
   - From a Powershell command-line, enter the executable name (`.\ad-passwordchange-handler-1.7.0.exe`), and press `Enter`.

   Tip

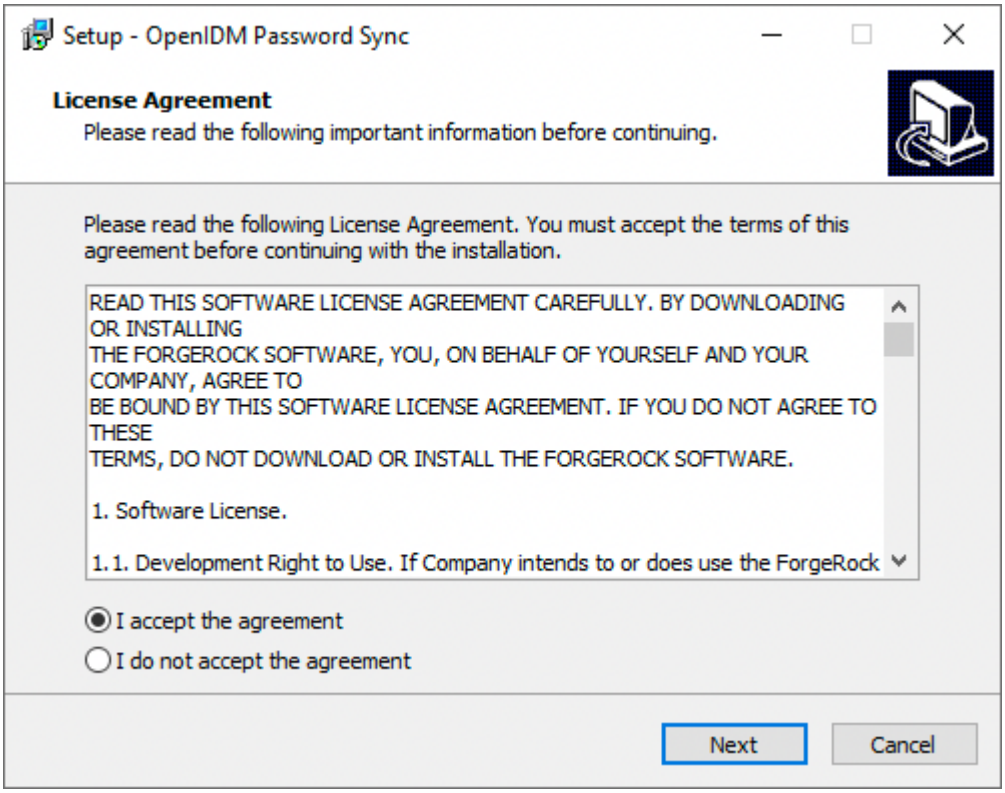
   When starting the installation wizard from the command-line, the following options are available:
   - To save the settings in a configuration file, use the `/saveinf` switch:

     ```
     PS C:\path\to\dir> .\ad-passwordchange-handler-1.7.0.exe /saveinf=C:\temp\adsync.inf
     ```

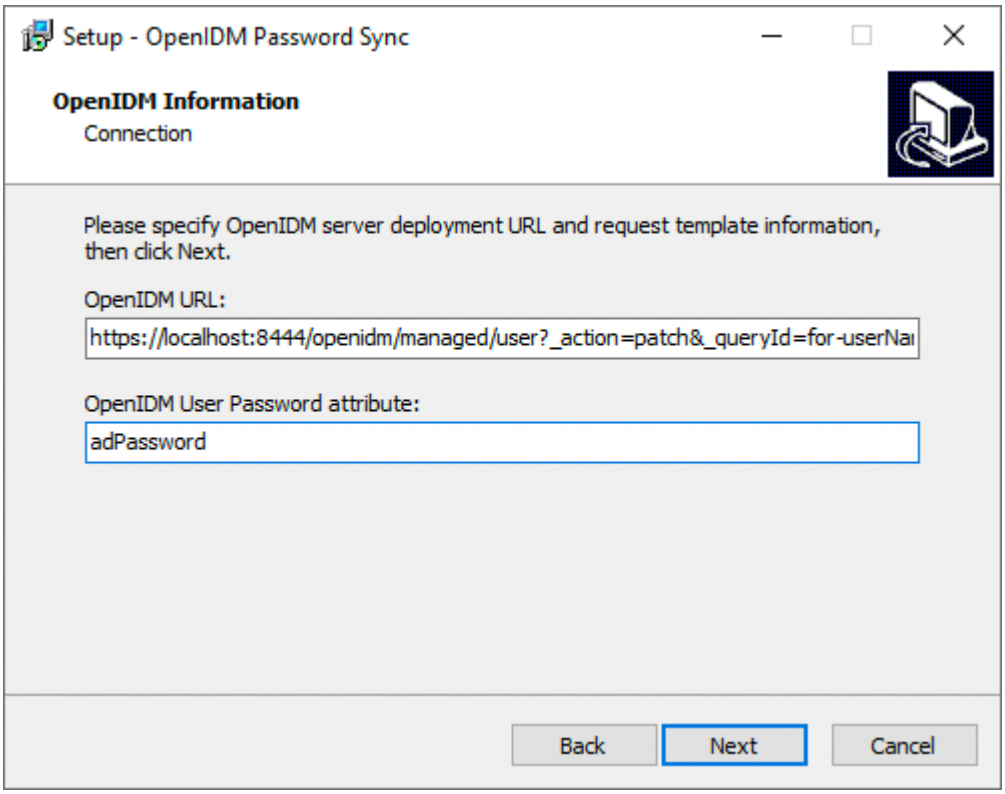
   - If you have a configuration file with installation parameters, you can install the password plugin in silent mode as follows:

     ```
     PS C:\path\to\dir> .\ad-passwordchange-handler-1.7.0.exe /verysilent /loadinf=C:
     \temp\adsync.inf
     ```

3. In the **Setup - OpenIDM Password Sync** window, on the **License Agreement** page, you must accept the license agreement to continue, and then click **Next**.

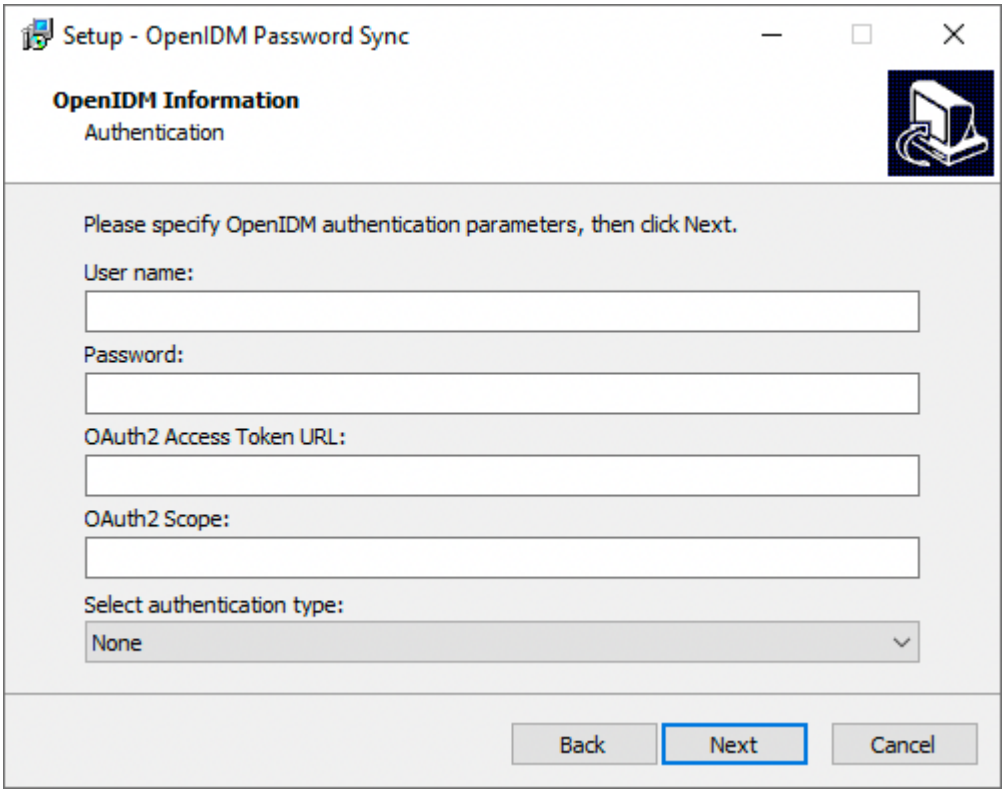


4. On the **OpenIDM Information: Connection** page, enter the applicable information in the following fields, and click **Next**.



OpenIDM URL	The URL where IDM is deployed, including the query that targets each user account. For example: `https://localhost:8444/openidm/managed/user?_action=patch&_queryFilter=userName+eq+'${samaccountname}'`
OpenIDM User Password attribute	The password attribute for the `managed/user` object, such as `adPassword`. **Tip** If the `password` attribute does not exist in the IDM `managed/user` object, the password sync service will return an error when it attempts to replay a password update that has been made in Active Directory. If your managed user objects do not include passwords, you can add an `onCreate` script to the **Active Directory > Managed Users** mapping that sets an empty password when managed user accounts are created. The following excerpt of a sample `sync.json` file shows such a script in the mapping: `"mappings" : [ { "name" : "systemAdAccounts_managedUser", "source" : "system/ad/account", "target" : "managed/user", "properties" : [ { "source" : "sAMAccountName", "target" : "userName" } ], "onCreate" : { "type" : "text/javascript", "source" : "target.password=''" }, ... } ]` The onCreate script creates an empty password in the `managed/user` object, so that the password attribute exists and can be patched.

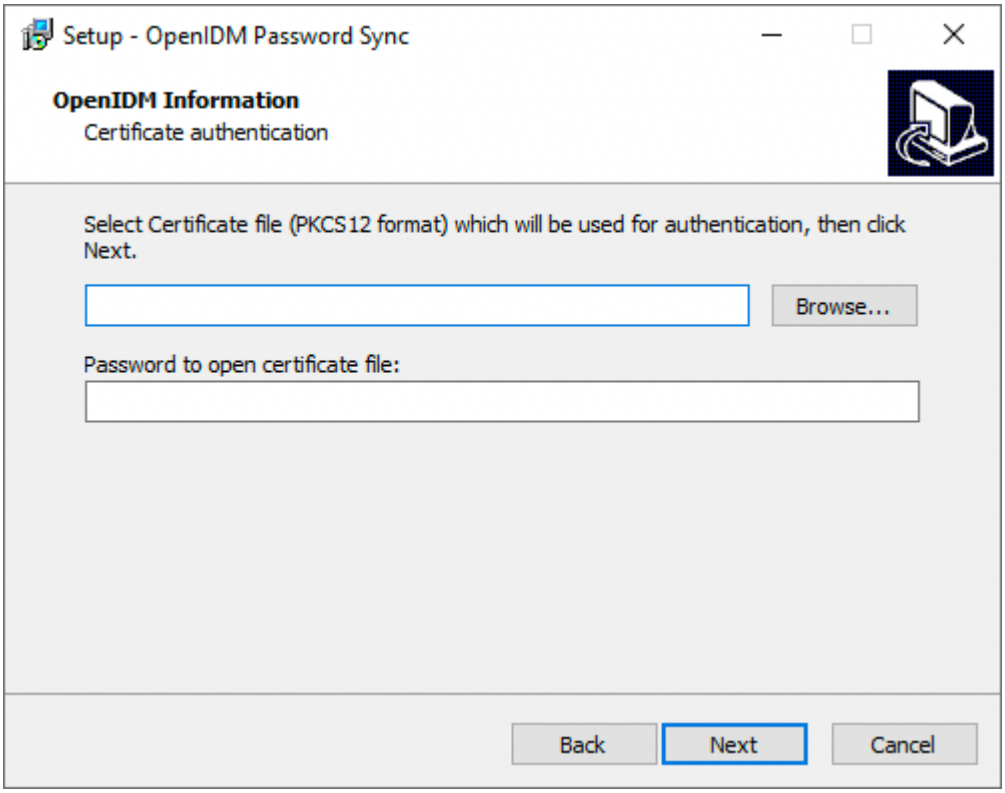
5. On the **OpenIDM Information: Authentication** page, enter the applicable information in the following fields, and click **Next**.



User name	An administrative user that can authenticate to IDM. For example, `openidm-admin`.
Password	The above, specified user's password.
OAuth2 Access Token URL	If you are using the authentication type `OAuth2 Access Token`, enter the token URL. For example: `https://am.example.com/am/oauth2/realms/root/access_token`
OAuth2 Scope	If you are using the authentication type `OAuth2 Access Token`, enter the OAuth2 token scope. For example `fr:idm:*`.
Select authentication type	Select the authentication type that Active Directory will use to authenticate to IDM: ◦ To use plain HTTP authentication, select **OpenIDM Header**. ◦ To use mutual SSL authentication, select **Certificate**. ◦ To use AM bearer tokens, select **OAuth2 Access Token**.

6. If you selected **Certificate** as the authentication type, complete this step; otherwise, skip to the next step.

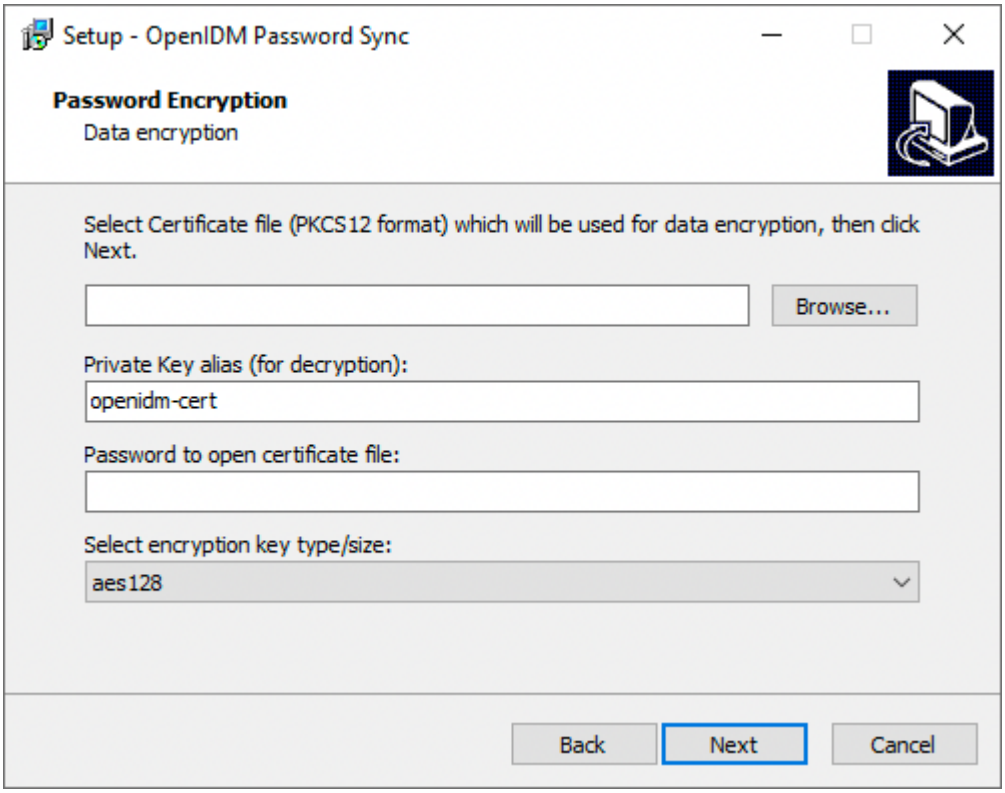
   On the **OpenIDM Information: Certificate Authentication** page, enter the applicable information in the following fields, and click **Next**.



Select Certificate file...	Browse to select the certificate file that Active Directory will use to authenticate to IDM.
	
	Important
	The certificate file must be configured with an appropriate encoding, cryptographic hash function, and digital signature. The password synchronization plugin can read a public or a private key from a PKCS #12 archive file.
	For production purposes, you should use a certificate that has been issued by a certificate authority. For testing purposes, you can generate a self-signed certificate.
	
	You must also import the certificate into the IDM `keystore.jceks` file. On the machine that is running IDM, enter the following command:
	
	`keytool \`
	`-importkeystore \`
	`-srckeystore /path/to/ad-pwd-plugin-localhost.p12 \`
	`-srcstoretype PKCS12 \`
	`-destkeystore keystore.jceks \`
	`-deststoretype JCEKS`

Private key alias	The certificate alias, such as `ad-pwd-plugin-localhost`. The password sync plugin sends the alias when communicating with IDM, which uses the alias to retrieve the corresponding private key in IDM's keystore. Update the IDM secret store (`conf/secrets.json`) to add this certificate alias to the `idm.default` secretId: `{ "stores" : [{ "name" : "mainKeyStore", "class" : "org.forgerock.openidm.secrets.config.FileBasedStore", "config" : { ... "mappings" : [{ "secretId" : "idm.default", "types" : ["ENCRYPT", "DECRYPT"], "aliases" : ["&{openidm.config.crypto.alias|openidm-sym-default}", "ad-pwd-plugin-localhost"] }, ...] } }, ...], ... }` For more information about secret stores, refer to Secret stores.
Password to open certificate file	The keystore password (`changeit`, in the previous example).

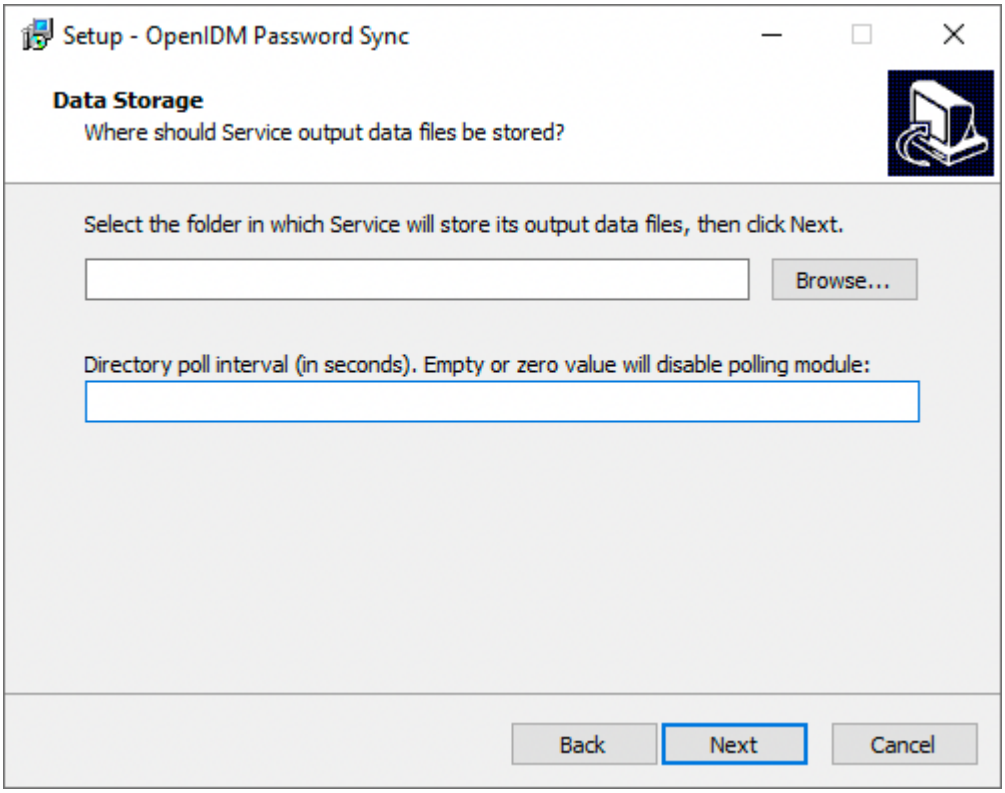
7. On the **Password Encryption** page, enter the applicable information in the following fields, and click **Next**.



Select Certificate file...	Browse to select the certificate that will be used for password encryption. The certificate format must be PKCS #12.
	
	Important
	The certificate file must be configured with an appropriate encoding, cryptographic hash function, and digital signature. The plugin can read a public or a private key from a PKCS #12 archive file.
	For production purposes, you should use a certificate that has been issued by a certificate authority. For testing purposes, you can generate a self-signed certificate.
	
	You must also import the certificate into the IDM truststore. On the machine that is running IDM, enter the following command:
	
	`keytool \`
	`-importkeystore \`
	`-srckeystore /path/to/ad-pwd-plugin-localhost.p12 \`
	`-srcstoretype PKCS12 \`
	`-destkeystore truststore \`
	`-deststoretype JKS`

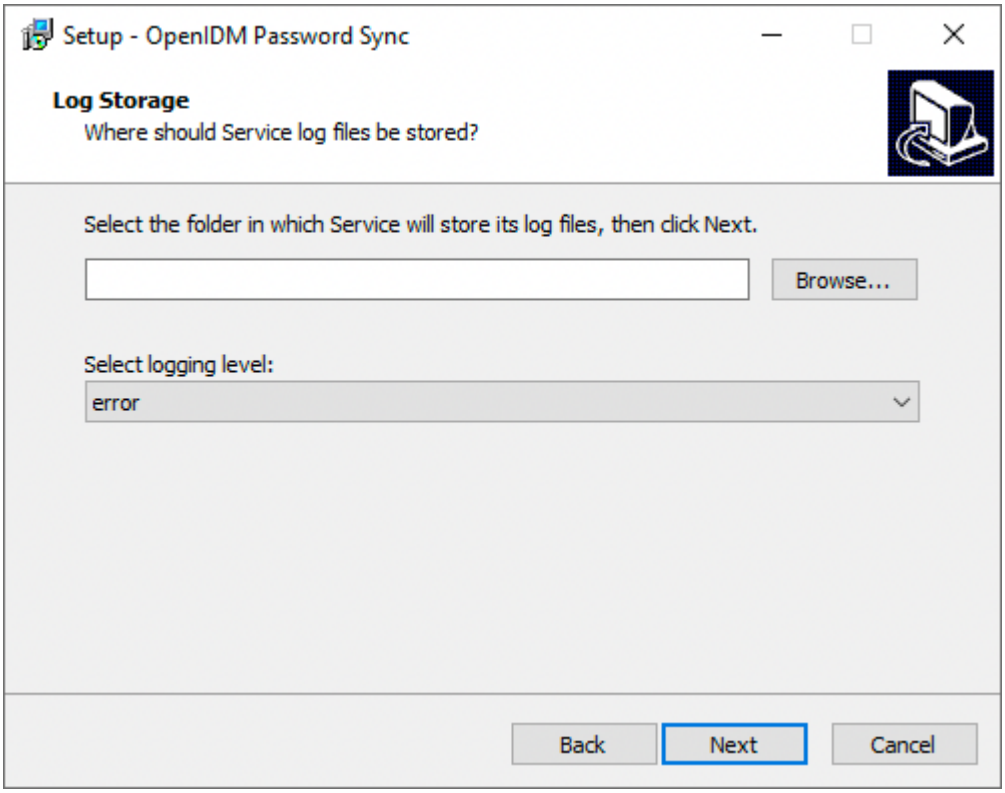
Private key alias	The certificate alias, such as `ad-pwd-plugin-localhost`. The password sync plugin sends the alias when communicating with IDM, which uses the alias to retrieve the corresponding private key in IDM's keystore. Update the IDM secret store (`conf/secrets.json`) to add this certificate alias to the `idm.default` secretId: `"mappings": [ { "secretId": "idm.default", "types": [ "ENCRYPT", "DECRYPT" ], "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}", "ad-pwd-plugin-localhost" ] }, ... } ]` For more information about secret stores, refer to Secret stores.
Password to open certificate file	The password to access the PFX keystore file, such as `changeit`, from the previous example.
Select encryption...	The encryption standard to use when encrypting the password value (AES-128, AES-192, or AES-256).

8. On the **Data Storage** page, enter the applicable information in the following fields, and click **Next**.



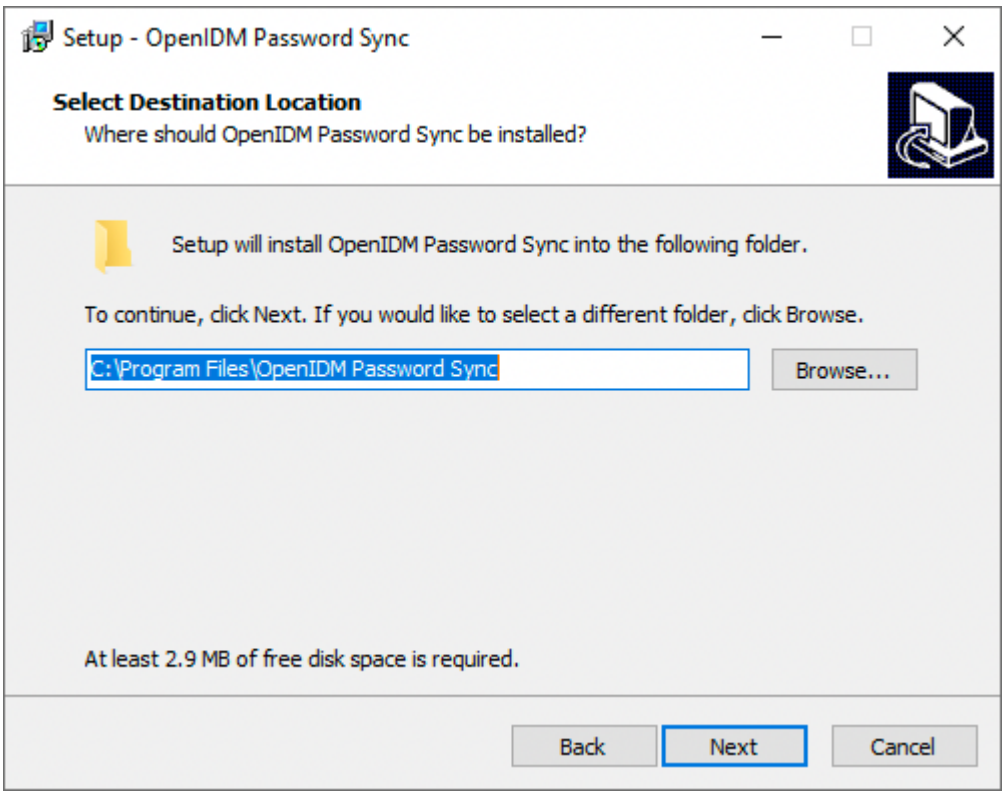
Select the folder in which Service will store its output data files	Browse to select the folder for data output files. The server should prevent access to this folder except for the `Password Sync service`. **Important** The path name cannot include spaces.
Directory poll interval (seconds)	The number of seconds between calls to check whether IDM is available. For example, `60`, to poll IDM every minute.

9. On the **Log Storage** page, enter the applicable information in the following fields, and click **Next**.

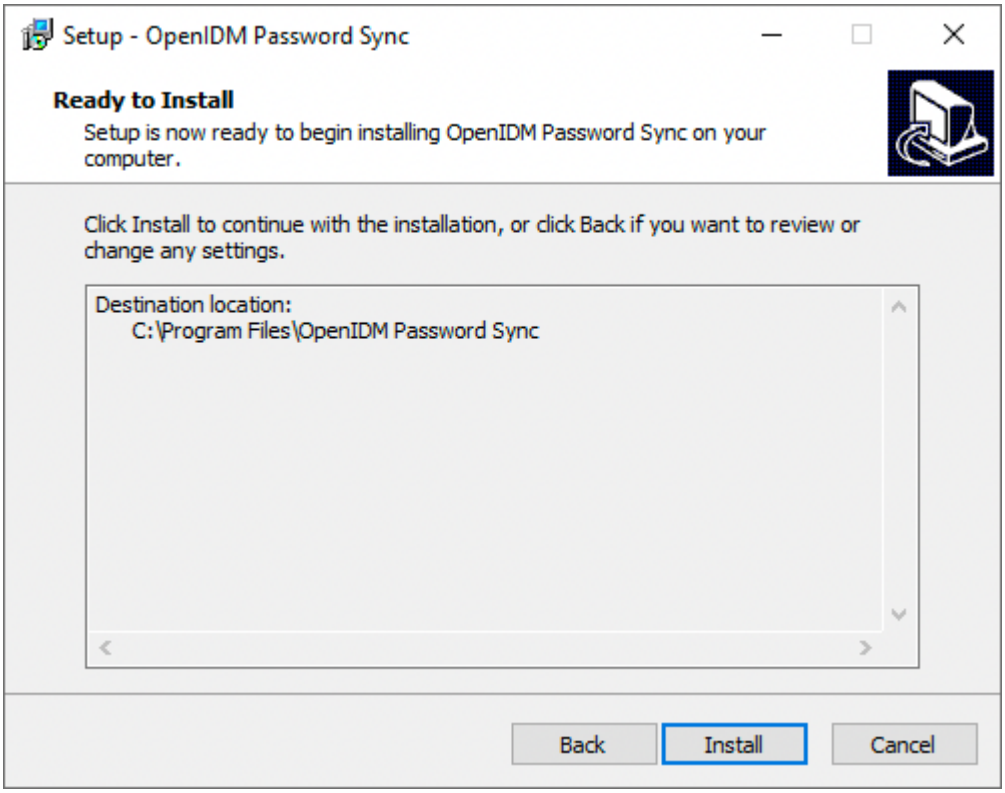


Select the folder in which Service will store its log files	Browse to select the folder for log files. **Important** The path name cannot include spaces.
Select logging level	The severity of messages to log: `error`, `info`, `warning`, `fatal`, or `debug`.

10. On the **Select Destination Location** page, browse to select the installation folder (default `C:\Program Files\OpenIDM Password Sync`), and click **Next**.



11. On the **Ready to Install** page, verify the details are acceptable, and click **Install** to continue. If you need to change any installation options, click **Back**.



12. On the **Completing the OpenIDM Password Sync Setup Wizard** page, select one of the following, and click **Finish**:

- **Yes, restart the computer now**
- **No, I will restart the computer later**

> **Important**
>
> If you select this option, you must restart the computer before you continue.

13. If you selected `Certificate` as the authentication type during setup, complete Add a Certificate to the Windows Certificate Store; otherwise, your setup is now complete.

Password synchronization should now be configured and working. To test that the setup was successful, change a user password in Active Directory. That password should be synchronized to the corresponding IDM managed user account, and you should be able to query the user's own entry in IDM using the new password.

Generate a self-signed certificate

For production purposes, you should use a certificate that has been issued by a certificate authority. For testing purposes, you can generate a self-signed certificate.

1. On the Active Directory host, generate a private key, which will be used to generate a self-signed certificate with the alias `ad-pwd-plugin-localhost`:

```
> keytool.exe ^
 -genkey ^
 -alias ad-pwd-plugin-localhost ^
 -keyalg rsa ^
 -dname "CN=localhost, O=AD-pwd-plugin Self-Signed Certificate" ^
 -keystore keystore.jceks ^
 -storetype JCEKS
Enter keystore password:changeit
Re-enter new password: changeit
Enter key password for <ad-pwd-plugin-localhost>
      <RETURN if same as keystore password>
```

2. Now use the private key, stored in the `keystore.jceks` file, to generate the self-signed certificate:

```
> keytool.exe ^
 -selfcert ^
 -alias ad-pwd-plugin-localhost ^
 -validity 365 ^
 -keystore keystore.jceks ^
 -storetype JCEKS ^
 -storepass changeit
```

3. Export the certificate. In this case, the `keytool` command exports the certificate in a PKCS #12 archive file format, used to store a private key with a certificate:

```
> keytool.exe ^
 -importkeystore ^
 -srckeystore keystore.jceks ^
 -srcstoretype jceks ^
 -srcstorepass changeit ^
 -srckeypass changeit ^
 -srcalias ad-pwd-plugin-localhost ^
 -destkeystore ad-pwd-plugin-localhost.p12 ^
 -deststoretype PKCS12 ^
 -deststorepass changeit ^
 -destkeypass changeit ^
 -destalias ad-pwd-plugin-localhost ^
 -noprompt
```

4. The PKCS #12 archive file is named `ad-pwd-plugin-localhost.p12`. Import the contents of the keystore contained in this file to the system that hosts IDM. To do so, import the PKCS #12 file into the IDM keystore file, named `truststore`, in the `/path/to/openidm/security` directory.

Add a certificate to the Windows certificate store

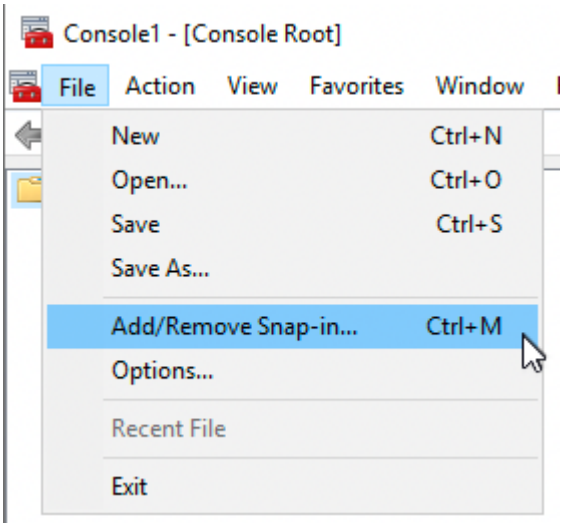
> **Note**
>
> If you selected `Certificate` as the authentication type during setup, you must complete this procedure.

The Password Sync Service uses Windows certificate stores to verify IDM's identity. The certificate that IDM uses must therefore be added to the list of trusted certificates on the Windows machine.

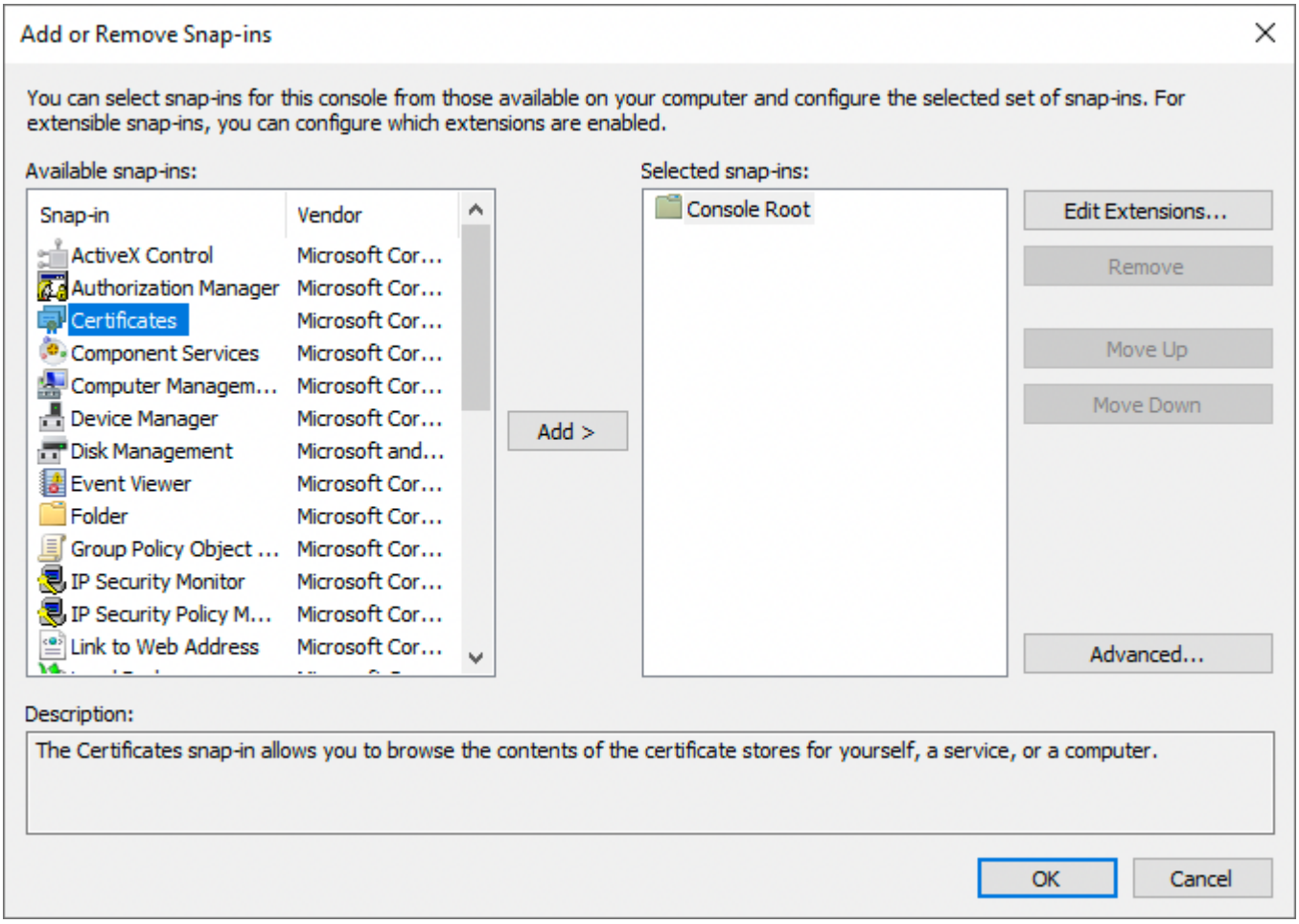
In a production environment, use a certificate that has been issued by a certificate authority. For test purposes, you can use the self-signed certificate that is generated by IDM on first startup.

To add the IDM certificate to the list of trusted certificates, use the **Microsoft Management Console**:

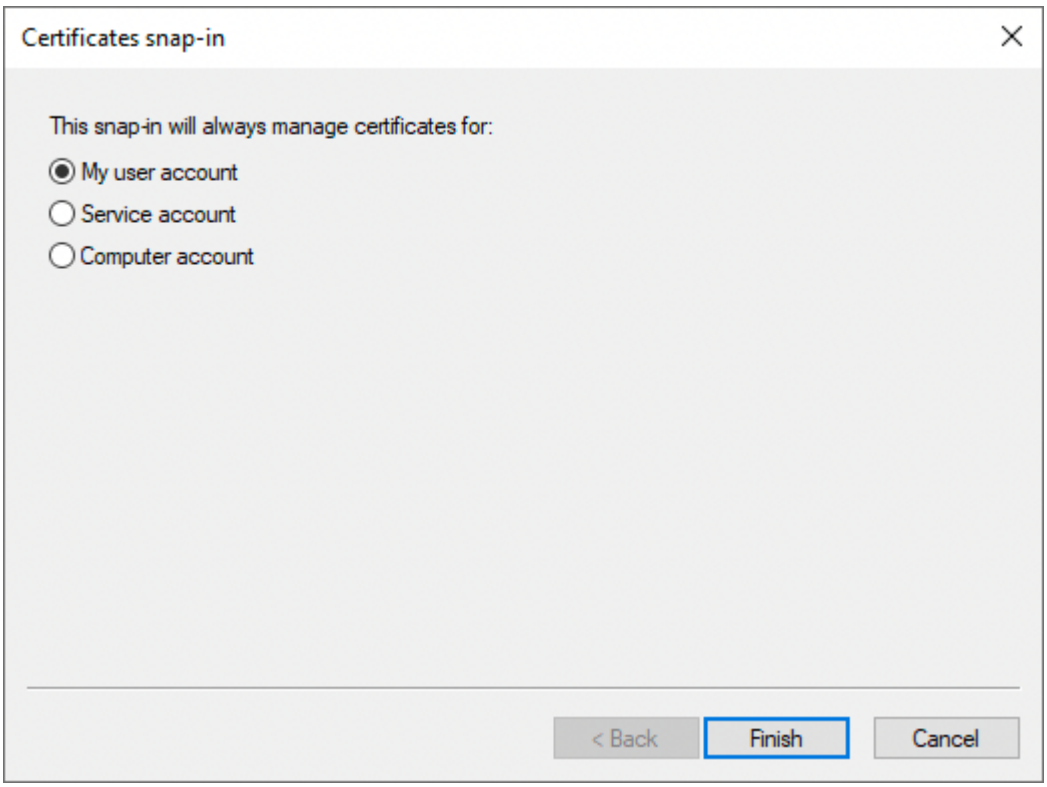
1. Click the Start Menu, type **mmc**, and click **mmc Run Command**.
2. In the Console window, select **File > Add/Remove Snap-in**.



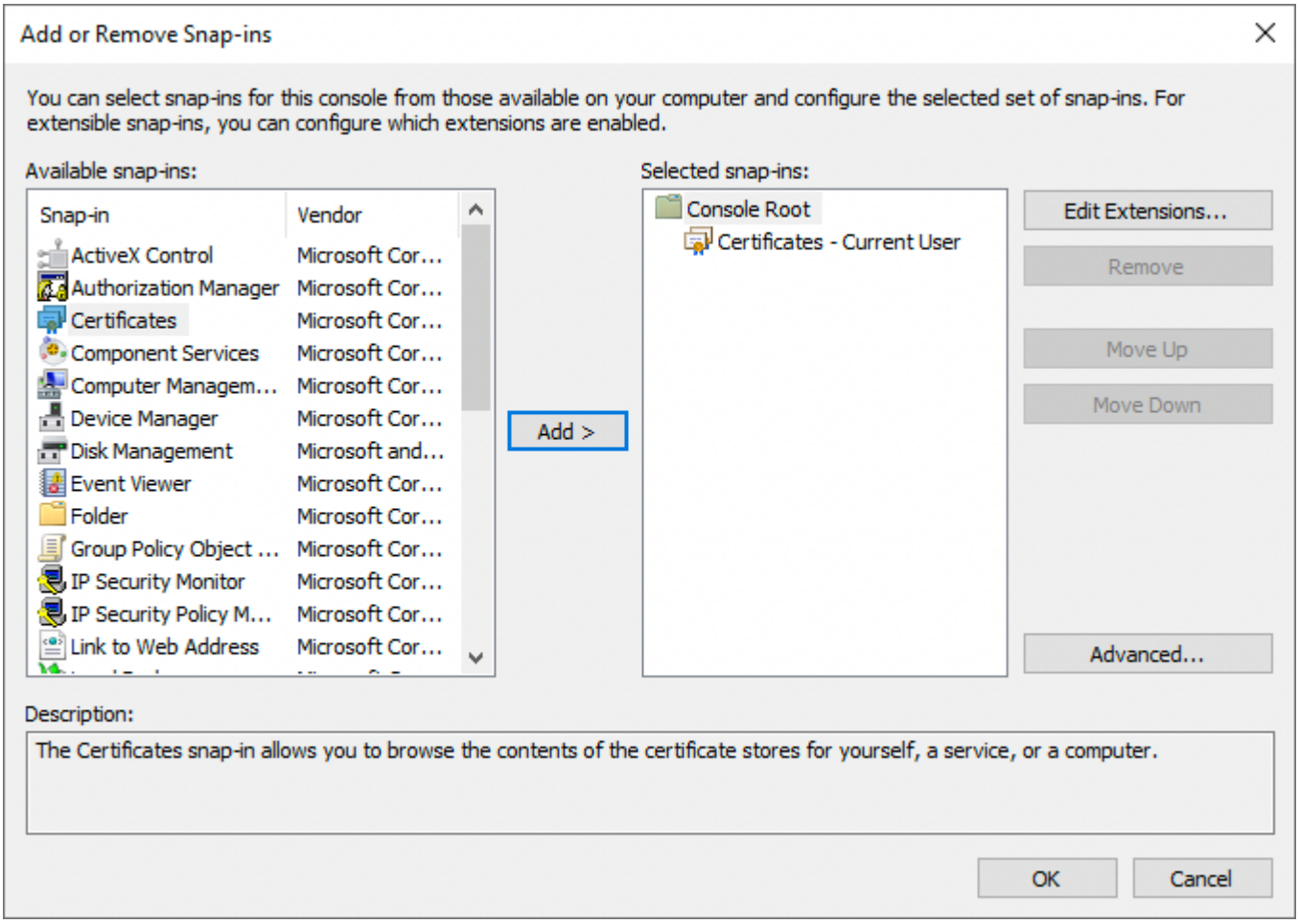
3. From the **Available snap-ins** area, select **Certificates**, and click **Add >**.



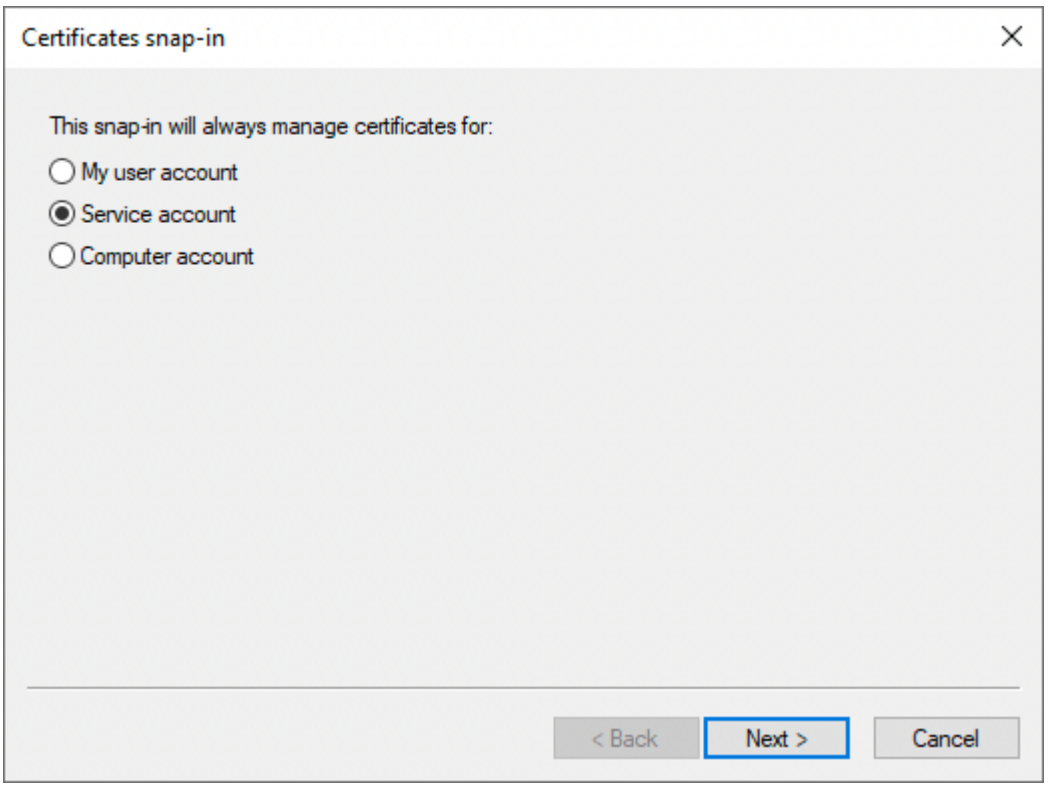
4. In the **Certificates snap-in** window, select **My user account**, and click **Finish**.



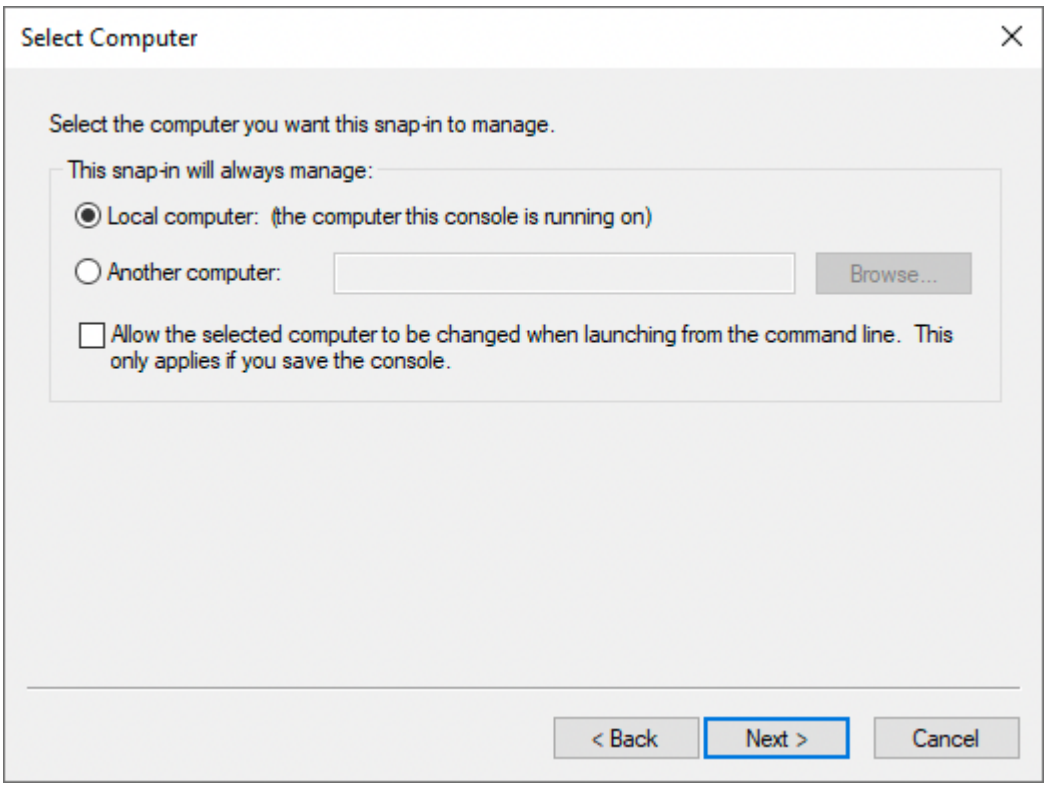
5. From the **Available snap-ins** area, select **Certificates**, and click **Add >**.



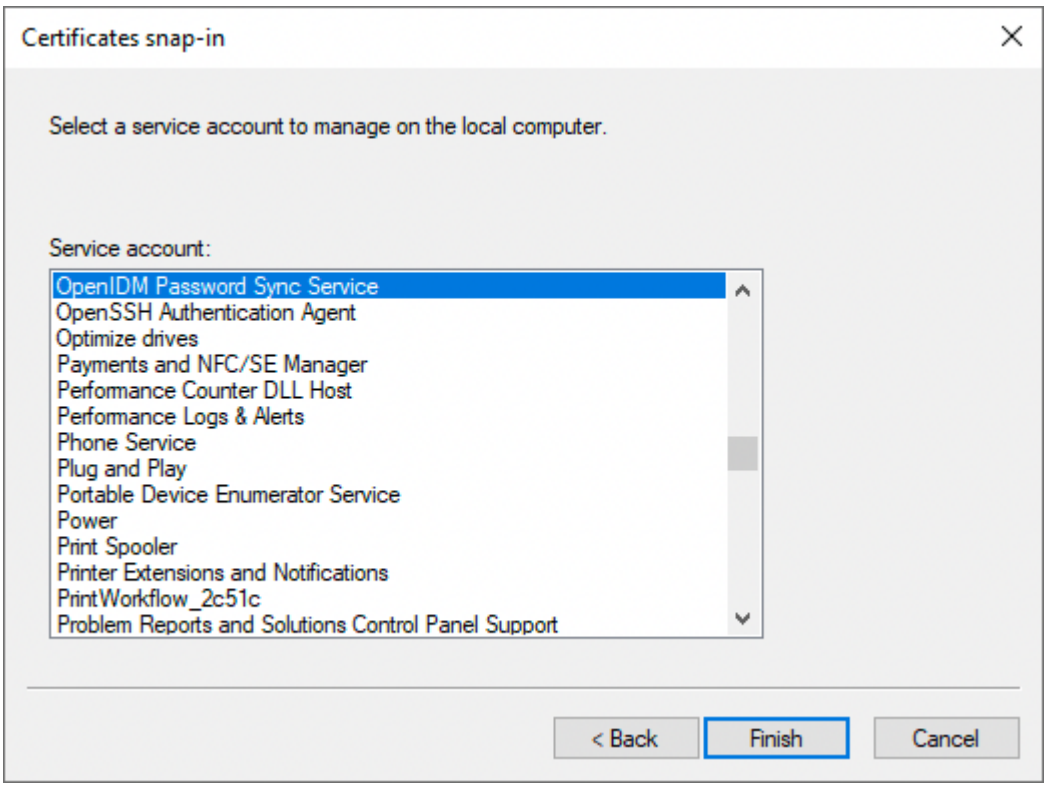
6. In the **Certificates snap-in** window, select **Service account**, and click **Next >**.



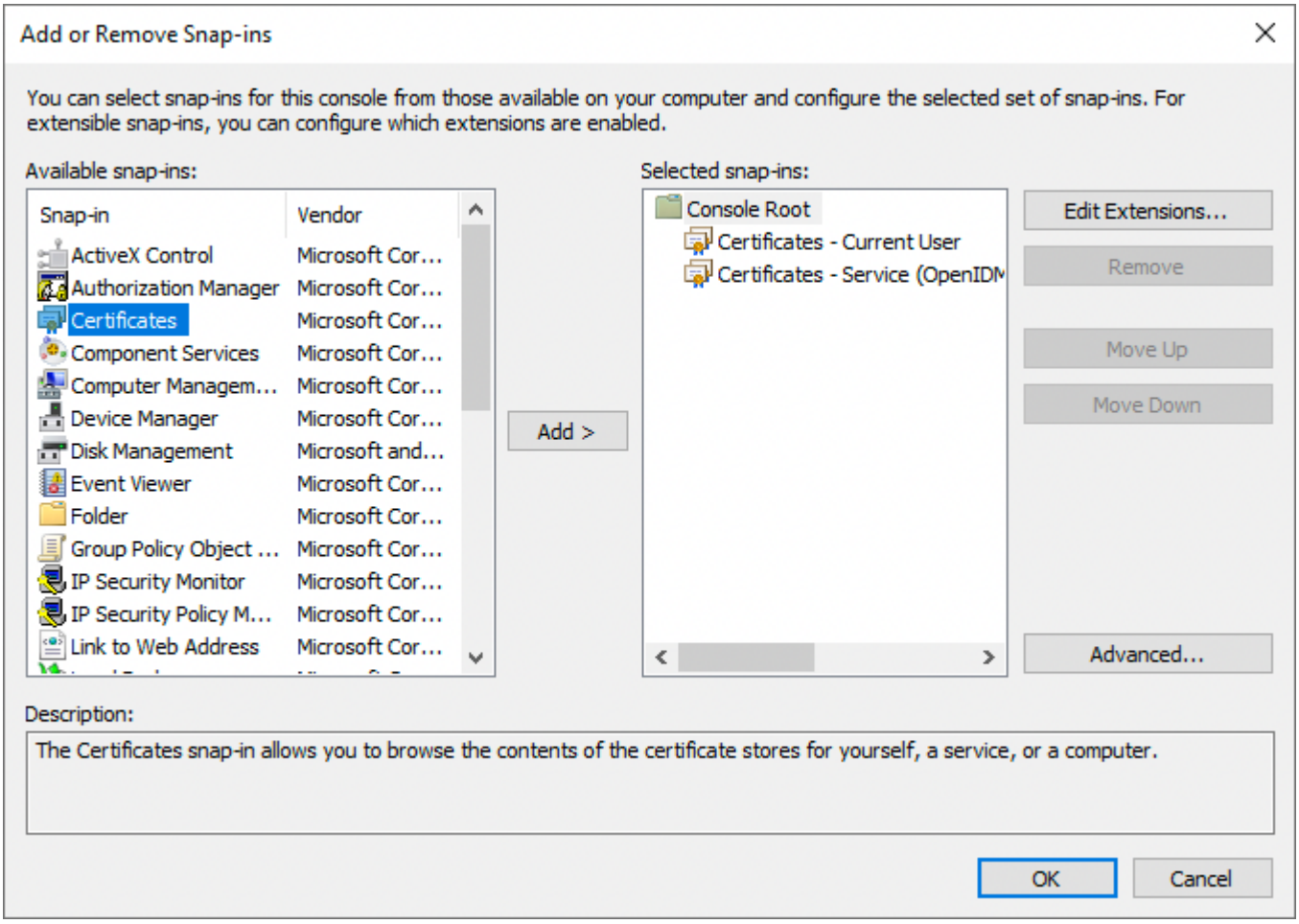
7. In the **Select Computer** window, select **Local Computer**, and click **Next >**.



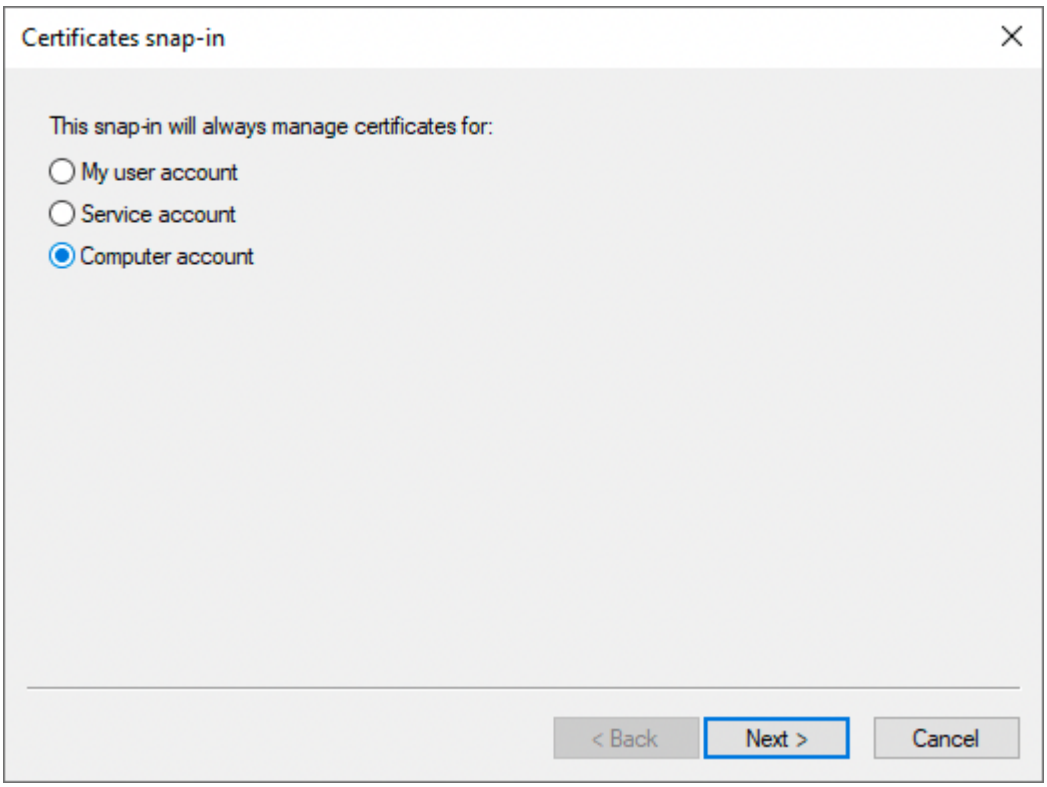
8. In the **Cerificates snap-in** window, select the service account **OpenIDM Password Sync Service**, and click **Finish**.



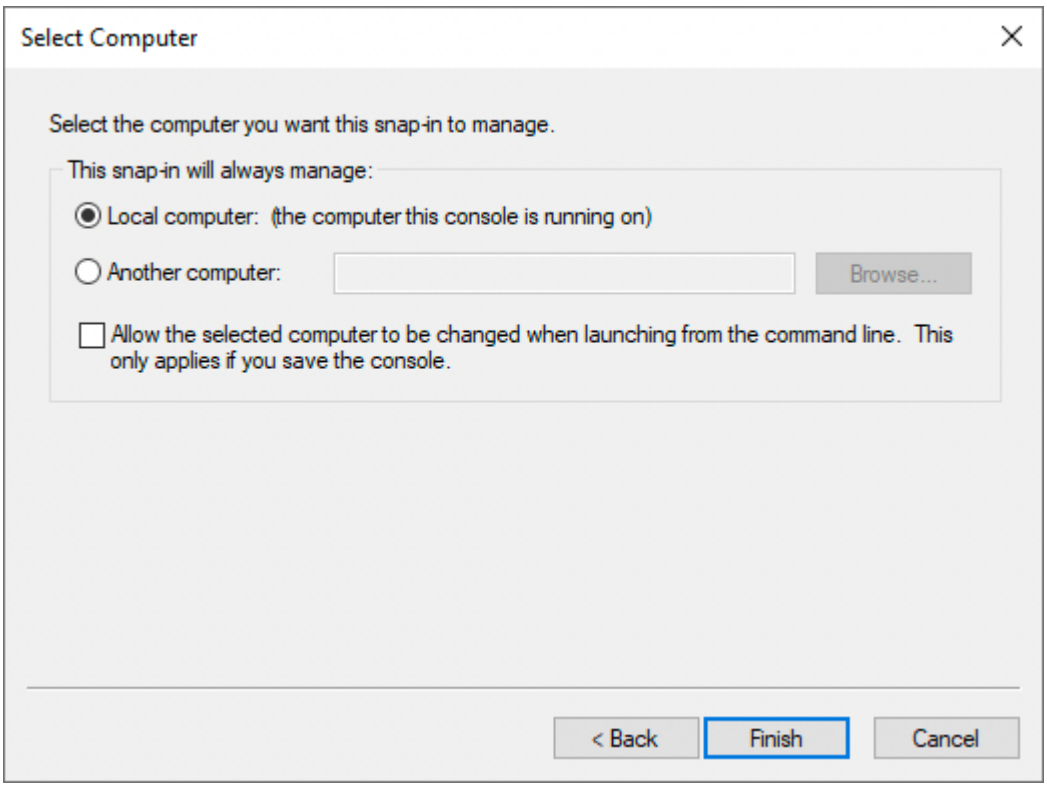
9. From the **Available snap-ins** area, select **Certificates**, and click **Add >**.



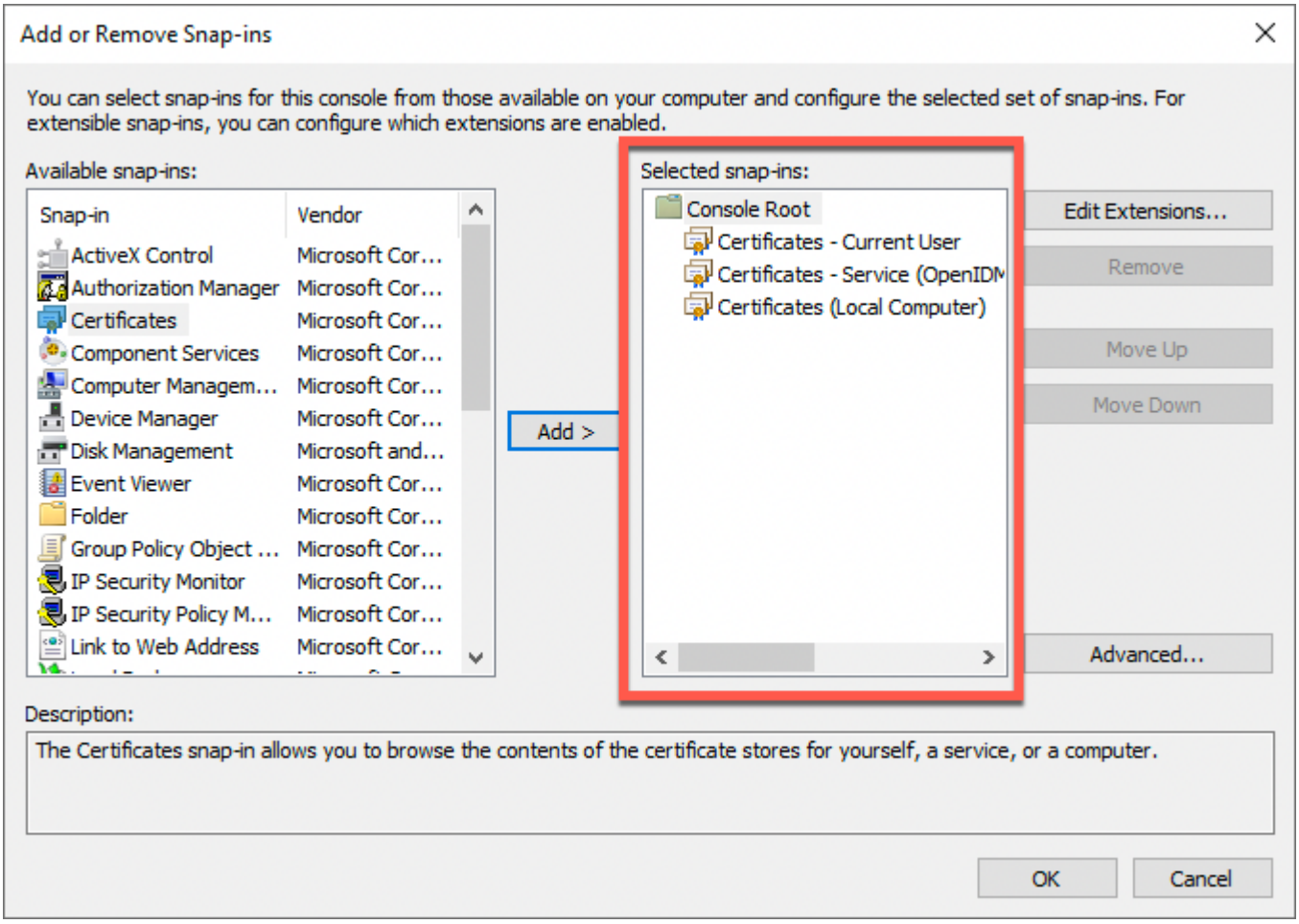
10. In the **Certificates snap-in** window, select **Computer account**, and click **Next >**.



11. In the **Select Computer** window, select **Local Computer**, and click **Finish**.



12. In the **Add or Remove Snap-ins** window, verify that you have three certificates in the **Selected snap-ins** area, and click **OK**. If you are missing any certificates, please review the earlier steps in this procedure, and add the missing certificate(s).

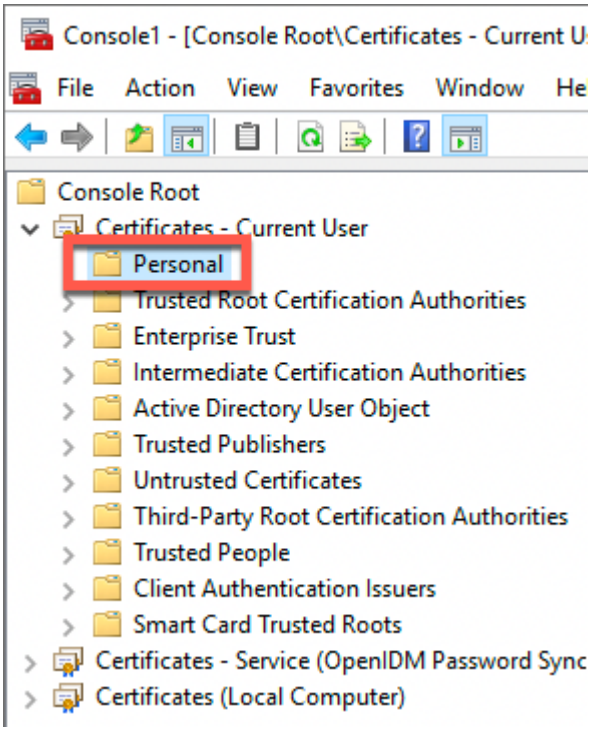


13. For each of the following nodes, import the certificate, as follows:

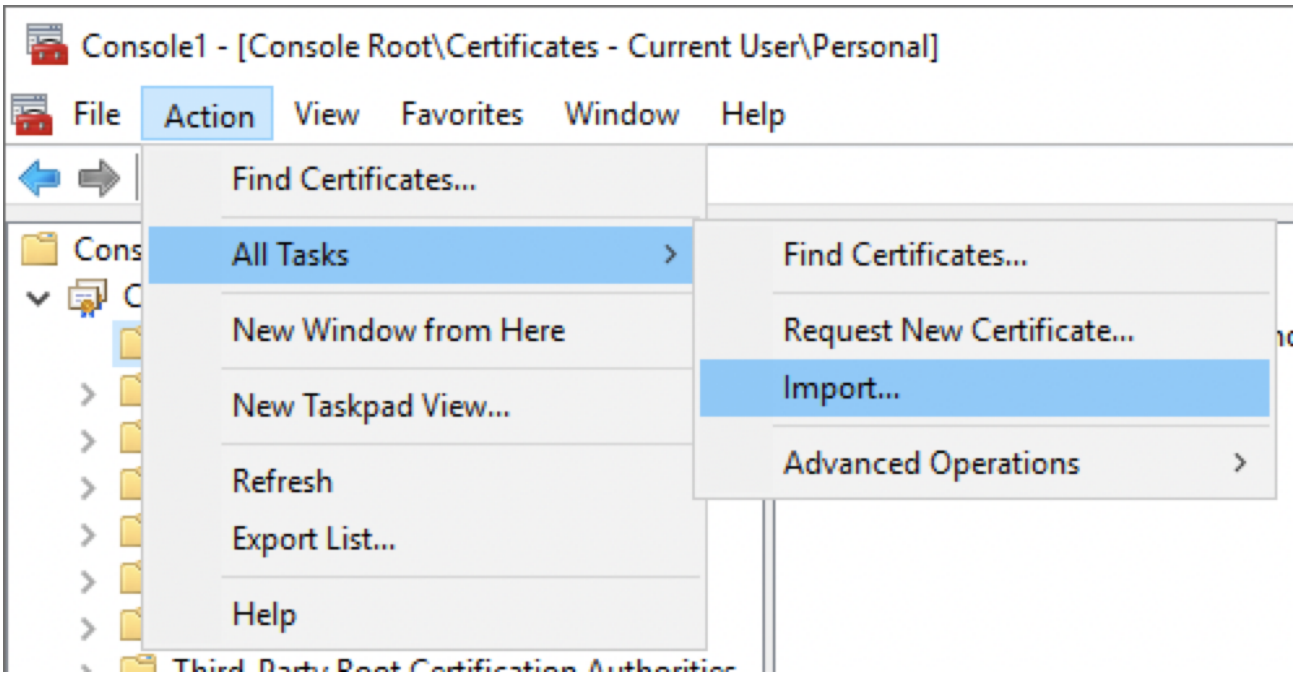
List of Nodes

Certificates - Current User > Personal
Certificates - Current User > Trusted Root Certification Authorities
Certificates - Service > OpenIDM Password Sync\Personal
Certificates - Service > OpenIDM Password Sync\Trusted Root Certification Authorities
Certificates > Local Computer > Personal
Certificates > Local Computer > Trusted Root Certification Authorities

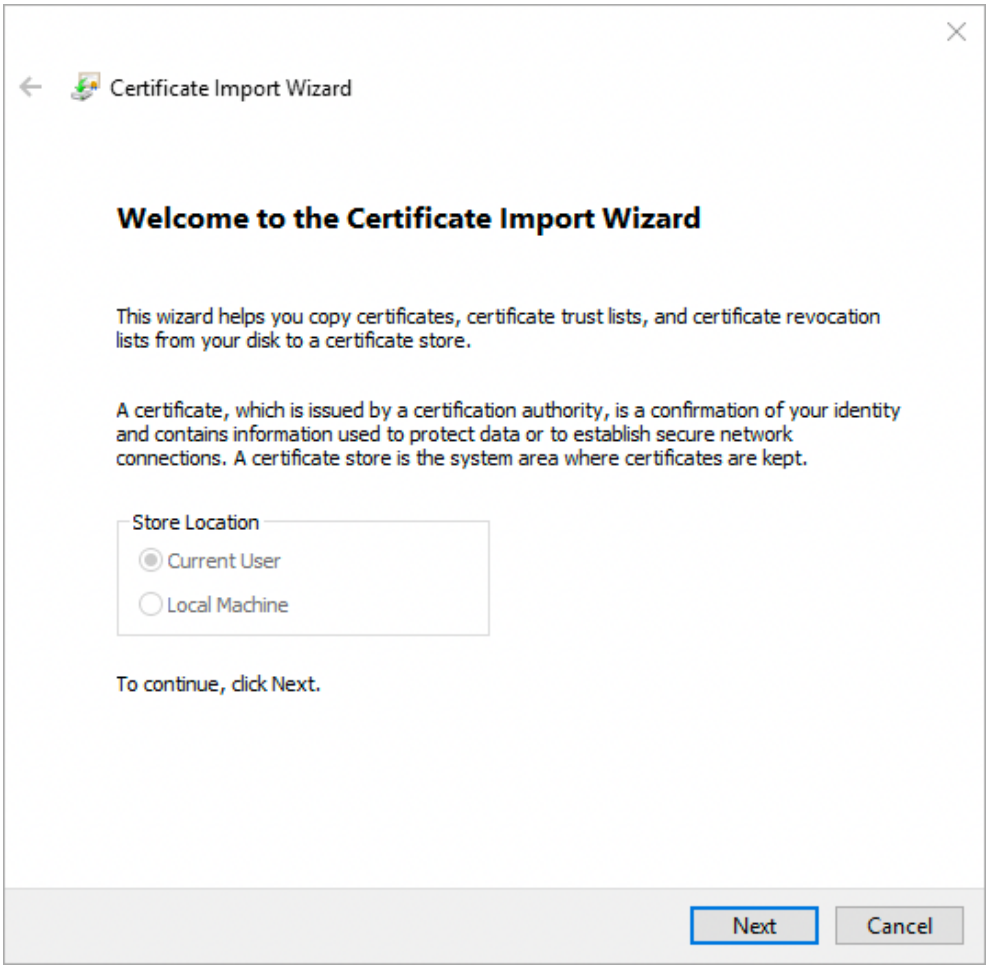
1. Expand the node, and select the appropriate item. For example, **Certificates - Current User > Personal**:



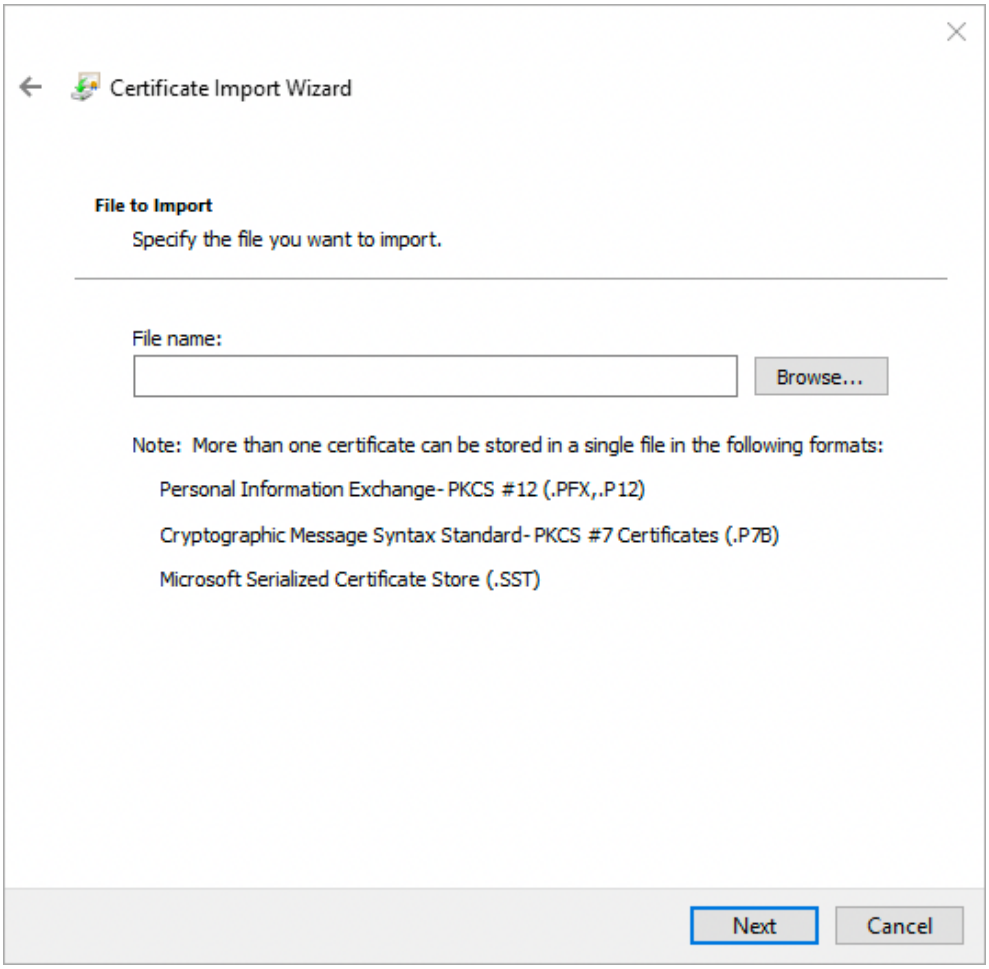
2. From the menu bar, select **Action** > **All Tasks** > **Import**.



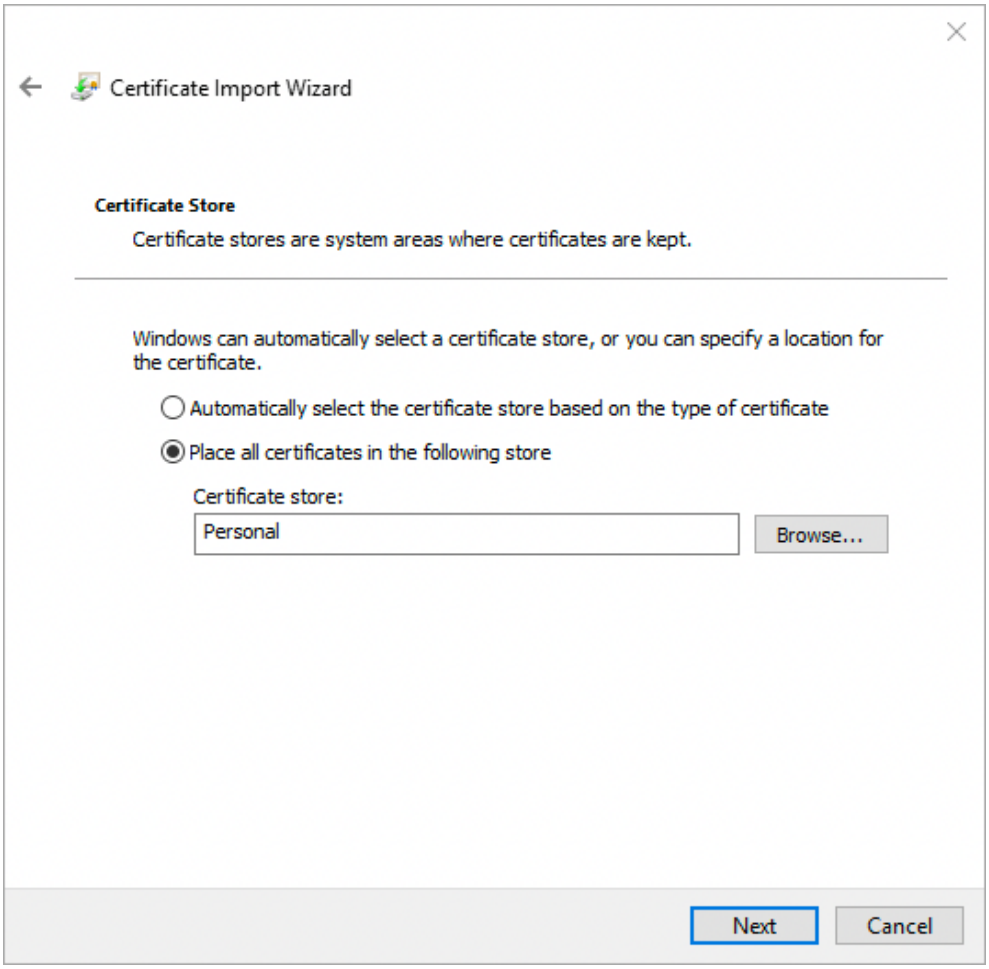
3. In the **Certificate Import Wizard** window, click **Next**.



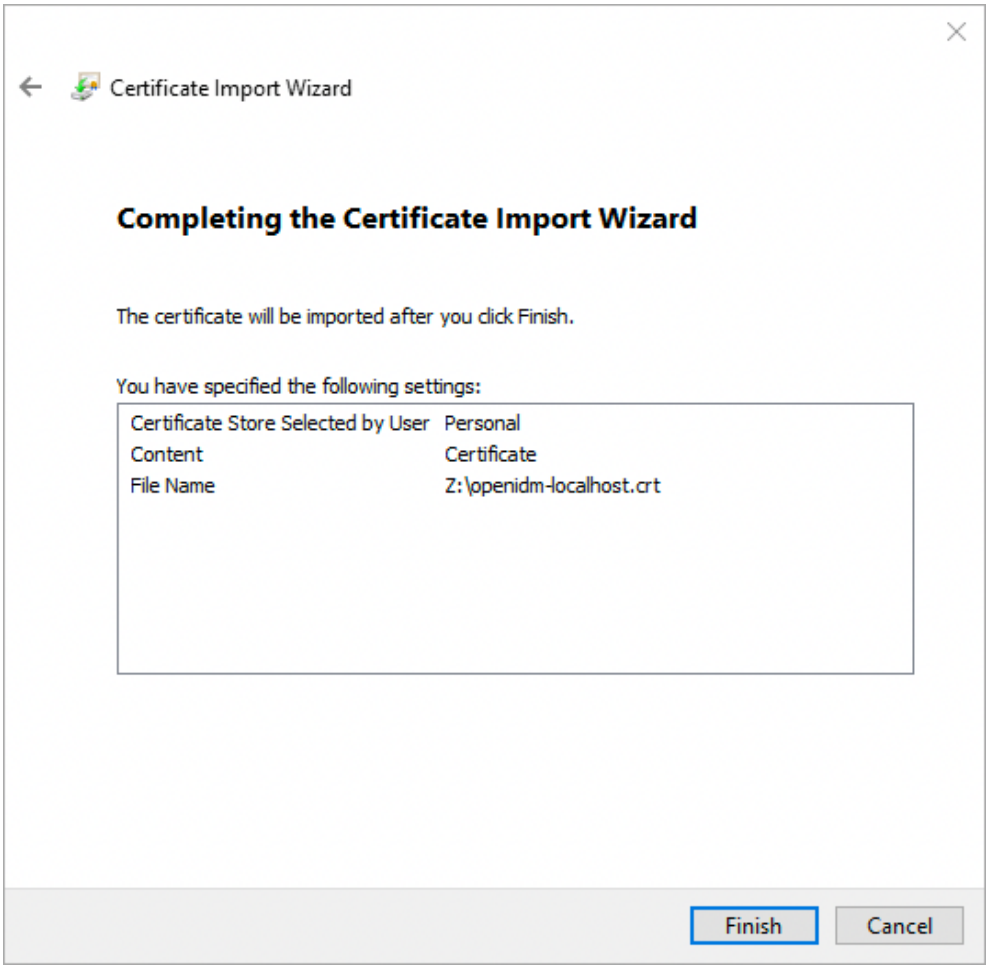
4. On the **File to Import** page, click **Browse**, locate the IDM certificate, and then click **Next**. If you exported IDM's self-signed certificate, the certificate is `openidm-localhost.crt`.



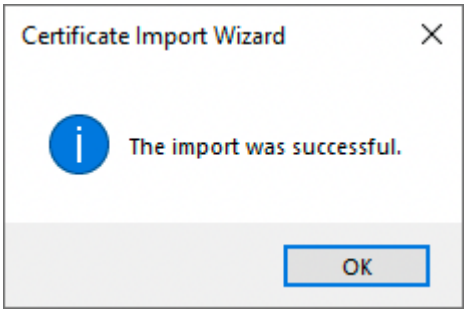
5. On the **Certificate Store** page, leave the default option selected, and click **Next**. For example:



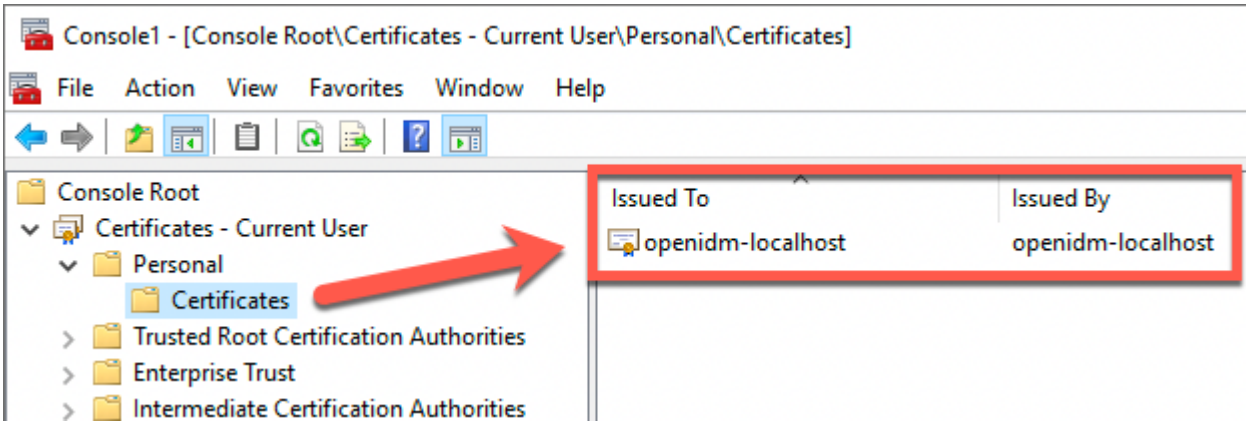
6. On the final page of the wizard, review the details, and click **Finish**. For example:



7. The **Certificate Import Wizard** window displays the success or failure of the import, click **OK**.



The main window of the **Microsoft Management Console** now displays the added certificate in a sub-node. For example:



> **Note**
>
> Make sure to repeat this sub-procedure for all required nodes.

Upgrade the Active Directory password synchronization plugin

To upgrade the Active Directory password synchronization plugin, you can run the installer over an existing configuration. For more information about installation, refer to Install the Active Directory password synchronization plugin.

Configure the Active Directory password synchronization plugin

If you need to change any settings after installation, access the settings using the Registry Editor. For the full list of available registry key values, refer to Registry Key Values. For information about creating registry keys, refer to the corresponding Windows documentation.

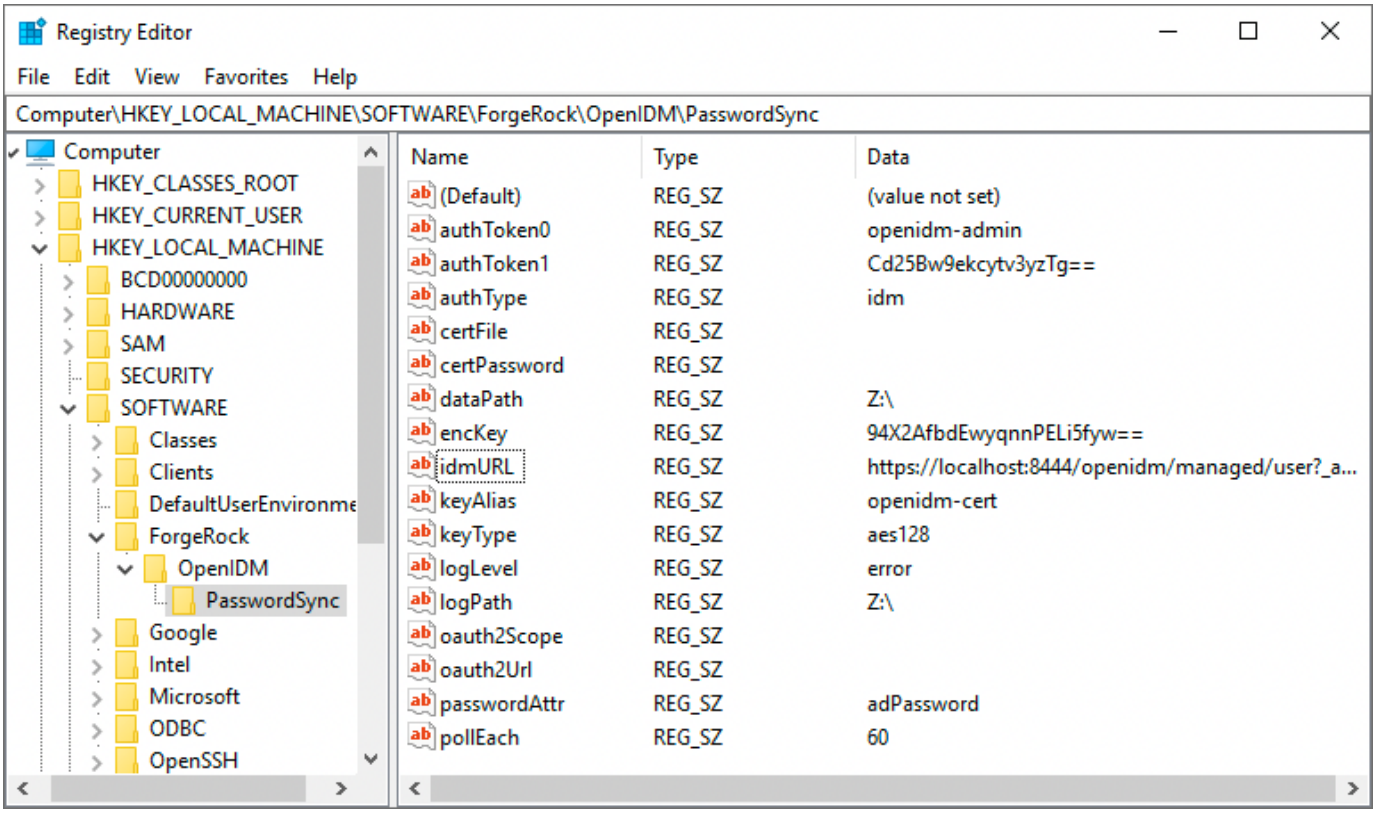
> **Note**
>
> After you change a registry key value associated with the password synchronization plugin, perform validation.

Create or edit registry key values

1. Click the Start Menu, type **registry**, and click **Registry Editor Desktop app**.
2. In the left pane of **Registry Editor**, expand the node:

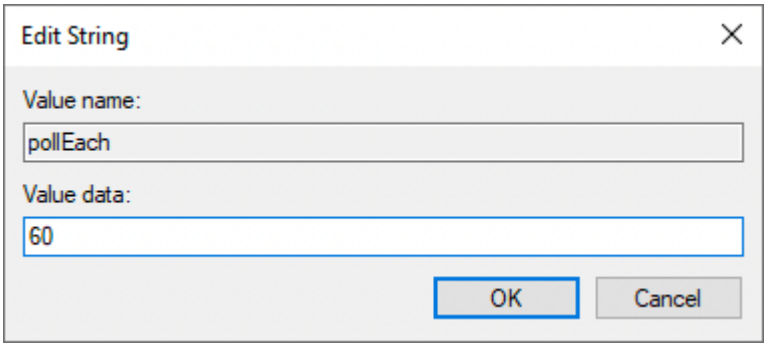
   ```
   HKEY_LOCAL_MACHINE > SOFTWARE > ForgeRock > OpenIDM > PasswordSync
   ```

   For Example:

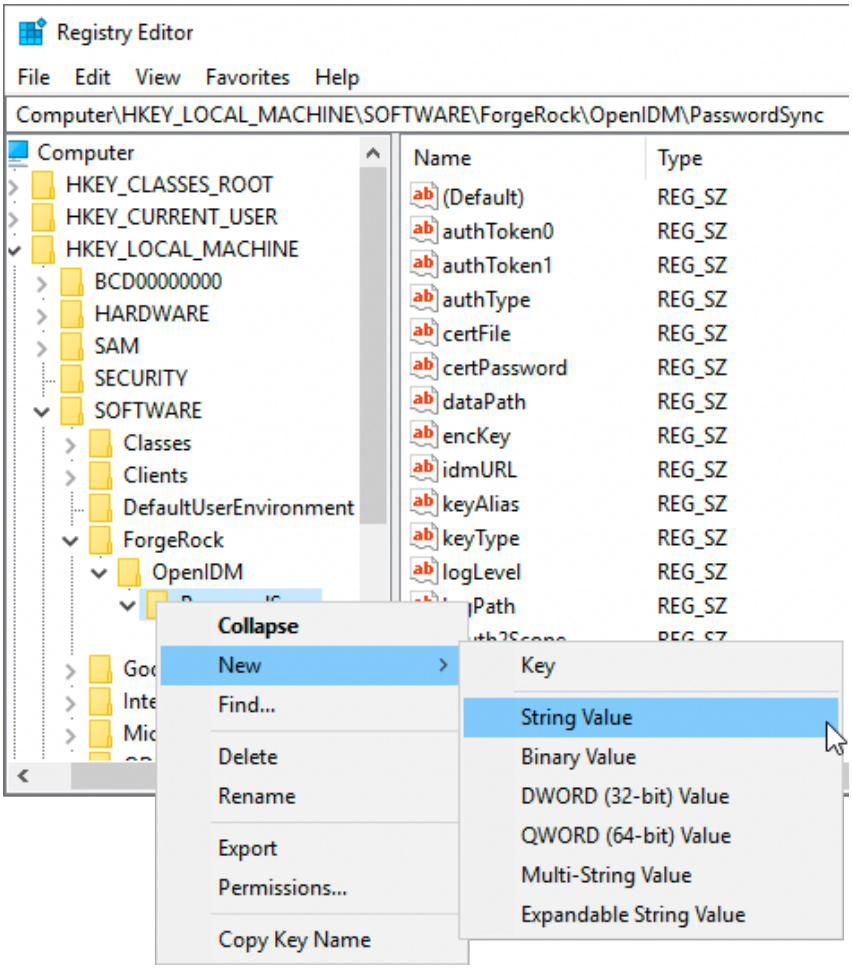


3. From here you can edit the registry key values, or create new ones.

   - To edit a value, double-click any item in the **Name** column. An **Edit String** window displays.



   - To create a new string value, right-click the last folder of the expanded node (**PasswordSync**), select **New > String Value**, and enter the applicable information.



Registry key values

Authentication method

The following registry key values let you customize the authorization method between the plugin and IDM.

Key Value	Description
`authType` (Required)	The authentication type: **`basic`** Plain HTTP or SSL authentication. **`idm`** Plain HTTP or SSL authentication using IDM headers. **`oauth2`** Oauth2 authentication using AM bearer tokens. **`cert`** Mutual SSL authentication using a certificate. **Note** By default, the plugin does not validate the IDM certificate. To enable validation, create the registry key value `netSslVerifyPeer` set to `True`.
`netSslVerifyPeer` (Optional)	When using `cert` as the authentication type, you can create this value set to `True` to force validation of the IDM certificate.
`authToken0` (Required)	The username or certificate path for authentication. For example, for plain HTTP or SSL authentication, `authToken0` might be set to `openidm-admin`. For certificate authentication, set `authToken0` to the certificate path. For example, `path/to/certificate/cert.p12`. Only PKCS #12 format certificates are supported.
`authToken1` (Required)	The authentication password. For example, for plain HTTP or SSL authentication, `authToken1` might be set to `openidm-admin`. For certificate authentication, set `authToken1` to the keystore password.
`encKey` (Optional)	The encryption key used to encrypt the values of `authToken1` and `certPassword`. These values are encrypted automatically when the plugin is installed, but when you change the settings, you can encrypt the values manually by setting the `encKey` registry key. For more information, refer to Registry Key Value Encryption. **Note** If you do not want to encrypt the values of `authToken1` and `certPassword`, you *must* remove the registry key value `encKey` from the registry.

Password encryption

The following registry key values let you customize the encryption method for captured passwords.

Key Value	Description
`certFile` (Required)	The path to the keystore used for encrypting captured passwords. For example, `path/to/keystore.p12`. Only PKCS #12 keystores are supported.
`certPassword` (Required)	The `certFile` keystore password.
`keyAlias` (Required)	The `certFile` keystore alias.
`keyType` (Optional)	The `certFile` keystore encryption algorithm. For example, `aes128`. If not set, defaults to `aes128`.

Connection & synchronization

The password synchronization plugin assumes that the Active Directory user attribute is `sAMAccountName`. Although the default attribute works in most deployments, you can specify an alternative attribute. For an example, refer to Example `userAttribute` Modification.

Key Value	Description
`userAttribute` (Optional)	The attribute that identifies the Active Directory user. The password synchronization plugin assumes that the Active Directory user attribute is `sAMAccountName`. Used with `userSearchFilter`.
`userSearchFilter` (Optional)	The search filter for locating Active Directory users. Can only be used if `userAttribute` is also specified (even if the value is the default attribute `sAMAccountName`). **Note** To use `userSearchFilter` with a special attribute such as `memberOf`, the filter only tests for immediate group memberships, and not for membership in the *primary group* (typically, `cn=Users` or `cn=Domain Users`) of your domain. The filter *does not* handle nested memberships. For example, *User A* is member of *Group A*, which is a member of *Group B*—although *User A* is technically a member of *Group B*, the filter will not interpret it as such.
`idmURL` (Required)	The URL where IDM is deployed, including the query that targets each user account.
`passwordAttr` (Required)	The password attribute for the `managed/user` object, such as `adPassword`.
`oauth2Url` (Optional)	For the authentication type `OAuth2 Access Token`, the token URL. For example: `https://am.example.com/am/oauth2/realms/root/access_token`

Key Value	Description
`oauth2Scope` (Optional)	For the authentication type `OAuth2 Access Token`, the OAuth2 token scope. For example `fr:idm:*`.
`userSearchBaseDn` (Optional)	The bind BaseDN used in the Active Directory user search. Use to set a different base dn for attribute search. Default value is derived from `defaultNamingContext`. Can only be used if `userAttribute` is also specified (even if the value is the default attribute `sAMAccountName`). Used with `userSearchFilter`.
`userSearchBindPass` (Optional)	The bind password used in the Active Directory user search. Can only be used if `userAttribute` is also specified (even if the value is the default attribute `sAMAccountName`). Used with `userSearchFilter`.
`userSearchBindUser` (Optional)	The bind user used in the Active Directory user search. Can only be used if `userAttribute` is also specified (even if the value is the default attribute `sAMAccountName`). Used with `userSearchFilter`.
`userSearchFilterStrict` (Optional)	Additional control for the behavior of `userSearchFilter`: • If set to `true`, requires `userSearchFilter` to return a value, which you can use to filter out Active Directory users/groups from being password-synced. • The default behavior is `false`, which results in `userSearchFilter` only being used to look up the Active Directory user attribute, and if it fails, password synchronization is still attempted with a default attribute.

IDM availability

When IDM is unavailable, or when an update fails, the password synchronization plugin stores the user password change in a JSON file on the Active Directory system and attempts to resend the password change at regular intervals.

You can modify this behaviour with the following registry key values:

Key Value	Description
`dataPath` (Required)	The location where the password synchronization plugin stores the unsent changes. When any unsent changes have been delivered successfully, files in this path are deleted. The plugin creates one file for each user. This means that if a user changes their password three times in a row, you will refer to only one file containing the last change.
`maxFileRetry` (Optional)	The maximum number of password change retry attempts after which the plugin stops attempting to send changes.
`netTimeout` (Optional)	The length of time (in milliseconds) before the plugin stops attempting a connection.
`pollEach` (Optional)	The interval (in seconds) between each attempt to send changes.

Logging configuration

Key Value	Description
`logPath` (Optional)	The path to the log file. **Important** If you change this parameter, you *must* restart the machine for the new setting to take effect. If you change the `logPath` and do not restart the machine, the service will write the logs to the new location, but the sync module will continue to write logs to the old location until the machine is restarted.
`logSize` (Optional)	The maximum log size (in Bytes) before the log is rotated. When the log file reaches this size, it is renamed `idm.log.0` and a new `idm.log` file is created.
`logLevel` (Optional)	The severity of messages to log: • `debug` • `info` • `warning` • `error` • `fatal`

Other

Key Value	Description
`pwdChangeInterval` (Optional)	Infinite password sync loop prevention. When Active Directory syncs passwords with IDM bidirectionally, it is possible to enter an infinite loop, where Active Directory and IDM are constantly updating the password and telling the other system to do the same. To help prevent this situation, you can set the `pwdChangeInterval` key to the number of seconds that must elapse between password updates. **Note** This feature requires AD Password Synchronization Plugin version 1.4.0 or later. Because version 1.4.0 can fail to make a secure connection with certain Windows versions, ForgeRock recommends using a later version.

Registry key value encryption

For security reasons, you should encrypt the values of the `authToken1` and `certPassword` keys. During password synchronization plugin installation, they are encrypted automatically. If you need to change the values, you can encrypt the values manually by setting the `encKey` registry key value.

> **Note**
>
> If you do not want to encrypt the values of the `authToken1` and `certPassword` keys, you *must* delete `encKey` from the registry. In this case, all password attributes can be set in cleartext (unencrypted).

To encrypt the values of the `authToken1` and `certPassword`:

1. Optionally, generate a new encryption key; otherwise, you can use the existing encryption key and skip this step.

```
PS C:\Program Files\OpenIDM Password Sync> .\idmsync.exe --key

keyValue
```

2. Encrypt the sensitive registry key values:

```
PS C:\Program Files\OpenIDM Password Sync> .\idmsync.exe --encrypt "keyValue" "authToken1Value"

authToken1-keyValue
```

```
PS C:\Program Files\OpenIDM Password Sync> .\idmsync.exe --encrypt "keyValue" "certPasswordValue"

certPasswordValue-keyValue
```

3. Replace the existing values of `encKey`, `authToken1` and `certPassword` keys with the generated values.

   For instructions on editing registry key values, refer to Create or Edit Registry Key Values.

Registry key value validation

After you change a registry key value associated with the password synchronization plugin, run `path\to\idmsync.exe --validate` to perform validation. For example:

```
PS C:\Program Files\OpenIDM Password Sync> .\idmsync.exe --validate

OpenIDM Password Sync Service

Validating configuration parameters as user "Administrator"

Logging parameters:
logPath:
   "Z:\" has read/write access permissions.

logLevel:
   "error" is a valid logLevel entry.

logSize:
   "" is not a valid logSize entry. Will use default 5120000 byte file size limit.

Service and data storage parameters:
dataPath:
   "Z:\" has read/write access permissions.

pollEach:
   "75" is a valid pollEach entry.

OpenIDM service parameters:
idmURL:
   "https://localhost:8444/openidm/managed/user?_action=patch&_queryId=for-userName&uid=${samaccountname}" is a valid
idmURL entry.

keyAlias:
   "openidm-cert" is a valid keyAlias entry.

passwordAttr:
   "adPassword" is a valid passwordAttr entry.

idm2Only:
   Service is configured to run with OpenIDM version 3.x or newer.

netTimeout:
   "" is not a valid netTimeout entry. Will use default 16 second network timeout.

authType, authToken0 and authToken1:
   Service is configured to use "OAuth2 Access Token" authentication
   "oauth2" is a valid authType entry.

Password encryption parameters:
certFile and certPassword:
   "Z:\openidm-localhost.crt" file is a valid entry.
```

Example `userAttribute` modification

The password synchronization plugin assumes that the Active Directory user attribute is `sAMAccountName`. The default attribute will work in most deployments. If you cannot use the `sAMAccountName` attribute to identify the Active Directory user, set the following registry keys on your Active Directory server, specifying an alternative attribute. The examples in the following table use the `employeeId` attribute instead of `sAMAccountName`.

For instructions on editing registry key values, refer to Create or Edit Registry Key Values.

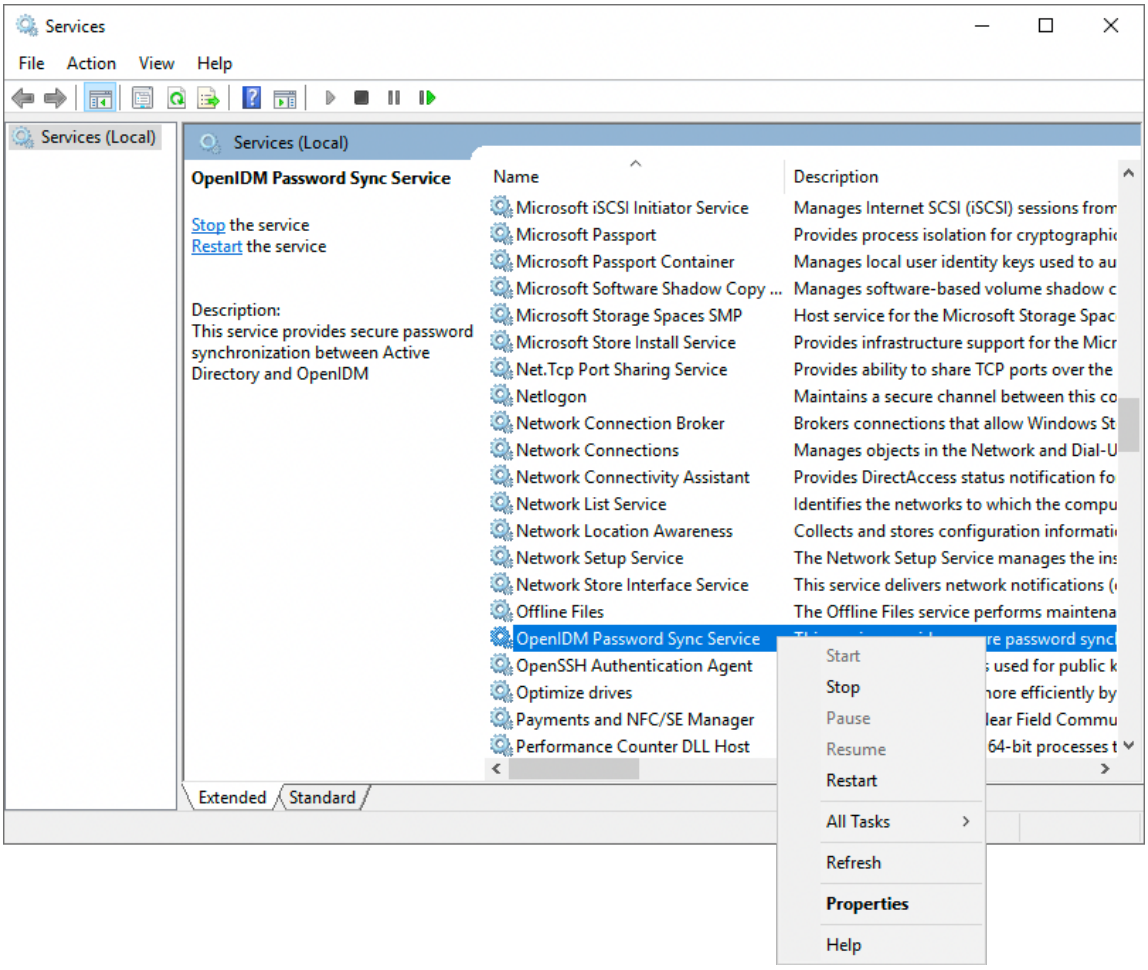
Key Value	Data
`userAttribute`	`employeeId`
`userSearchFilter`	`(&(objectClass=user)(sAMAccountName=%s))`
`idmURL`	`https://localhost:8444/openidm/managed/user?_action=patch&_queryFilter=uid+eq+${employeeId}`

Start or stop the plugin

The password synchronization plugin runs as **OpenIDM Password Sync Service**. You can start and stop the service using **Windows Service Manager** or the command line.

Windows Service Manager

1. Click the Start Menu, type **services.msc**, and click **Services Desktop App**.
2. In the **Services** windows, right-click **OpenIDM Password Sync Service**, and select the applicable option (**Start**, **Stop**, or **Restart**):



Command line

You can use the following commands to start and stop the service from the command line.

> **Note**
>
> The service will not start from the command line if any registry key value fails validation.

```
PS C:\Program Files\OpenIDM Password Sync> .\idmsync.exe --start
```

```
PS C:\Program Files\OpenIDM Password Sync> .\idmsync.exe --stop
```

Plugin status

You can also check the status of the plugin:

```
PS C:\Program Files\OpenIDM Password Sync> .\idmsync.exe --status
Service is running.
```

Help command

The password synchronization plugin executable file includes a help command that prints the available command line options.

```
PS C:\Program Files\OpenIDM Password Sync> .\idmsync.exe --help

OpenIDM Password Sync Service usage:

install service:
 idmsync.exe --install

uninstall service:
 idmsync.exe --remove

start service:
 idmsync.exe --start

stop service:
 idmsync.exe --stop

query service:
 idmsync.exe --status

validate configuration:
 idmsync.exe --validate

generate encryption key:
 idmsync.exe --key

encrypt password:
 idmsync.exe --encrypt "key" "password"

build and version info:
 idmsync.exe --version
```

Uninstall the Active Directory password synchronization plugin

You can uninstall the Active Directory Password Synchronization plugin from multiple locations:

- Uninstall from the Windows Control Panel (Control Panel > Programs and Features, select `OpenIDM Password Sync` from the list and select Uninstall).
- Run the uninstaller (`unins000.exe`) found in the OpenIDM Password Sync install directory (by default, `C:\Program Files\OpenIDM Password Sync`).

After the uninstaller finishes, Windows will prompt you to restart. Restart to complete the uninstall process.

Remove installed authentication certificates

If you selected to authenticate with mutual SSL authentication, you can manually remove the IDM certificates using the **Microsoft Management Console**:

1. Click the Start Menu, type **mmc**, and click **mmc Run Command**.
2. For each of the following nodes, expand the node, and delete the certificate.

List of Nodes

Certificates - Current User > Personal
Certificates - Current User > Trusted Root Certification Authorities
Certificates - Service > OpenIDM Password Sync\Personal
Certificates - Service > OpenIDM Password Sync\Trusted Root Certification Authorities
Certificates > Local Computer > Personal
Certificates > Local Computer > Trusted Root Certification Authorities

3. If the **OpenIDM Password Sync Service** is still listed with stored certificates:
    1. Select **File > Add/Remove Snap-in**.
    2. From the **Selected snap-ins** area, select **Certificates - OpenIDM Password Sync**, and click **Remove**.
    3. Click **OK**.

Audit

Guide to configuring audit logs and notifications.

Configure Audit

Configure audit logging.

Schema

Learn about the audit schema.

Notifications

Configure notifications.

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

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

Configure audit logging

The audit service publishes and logs information to one or more targets, including local data files, the repository, and remote systems.

Audit logs help you to record activity by account. With audit data, you can monitor logins, identify problems such as unresponsive devices, and collect information to comply with regulatory requirements.

The audit service logs information related to the following events:

- System access
- System activity

- Authentication operations
- Configuration changes
- Reconciliations
- Synchronizations

You can customize what is logged for each event type. Auditing provides the data for all relevant reports, including those related to orphan accounts.

When you first start IDM, an audit log file for each configured audit event topic is created in the `/path/to/openidm/audit` directory. Until there is a relevant event, these files will be empty.

When IDM sends data to these audit logs, you can query them over the REST interface.

Configure the audit service

You access the audit logging configuration over REST at the `openidm/config/audit` context path and in the `conf/audit.json` file. To configure the audit service, edit the `audit.json` file or use the admin UI. Select **Configure > System Preferences**, and click the **Audit** tab. The fields on that form correspond to the configuration parameters described in this section.

You can configure the following major options for the audit service:

Which audit handlers are used

Audit event handlers are responsible for handling audit events. They are listed in the `availableAuditEventHandlers` property in your `conf/audit.json` file.

Which handler is used for queries

You *must* configure one audit event handler to manage queries on the audit logs.

What events are logged

The events that are logged are configured in the `events` list for each audit event handler.

Track transactions across products

If you use more than one ForgeRock product, you can specify that a common `transactionId` be used to track audit data across products. Edit your `conf/system.properties` file and set:

```
org.forgerock.http.TrustTransactionHeader=true
```

Specify the audit query handler

By default, queries on audit logs are managed by the JSON audit event handler. You can configure one of the other available event handlers to handle queries. The audit event handler that you configure to manage queries must be `enabled`, either by including its definition in `audit.json`, or setting it to **Enabled** in the admin UI.

To specify which audit event handler should be used for queries, set the `handlerForQueries` property in the `audit.json` file, as follows:

```
{
    "auditServiceConfig" : {
        "handlerForQueries" : "json",
        "availableAuditEventHandlers" : [
            "org.forgerock.audit.handlers.csv.CsvAuditEventHandler",
            "org.forgerock.audit.handlers.jms.JmsAuditEventHandler",
            "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
            "org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHandler",
            "org.forgerock.openidm.audit.impl.RepositoryAuditEventHandler",
            "org.forgerock.openidm.audit.impl.RouterAuditEventHandler",
            "org.forgerock.audit.handlers.syslog.SyslogAuditEventHandler"
        ],
    ...
}
```

In this case, the `handlerForQueries` is set to `json`, which is the `name` of the `JsonAuditEventHandler`.

Important

- Do not use a file-based audit event handler, such as CSV or JSON, to handle queries *in a clustered environment*. ForgeRock recommends using an audit handler that aggregates audit records from all nodes in the cluster, such as JDBC.
  You can use a file-based audit handler for queries in a non-clustered demonstration or evaluation environment. However, be aware that these handlers do not implement paging, and are therefore subject to general query performance limitations.
- The JMS and Syslog handlers can *not* be used as the handler for queries.
- Logging via CSV or JSON may lead to errors in one or more mappings in the admin UI.

Choose audit event handlers

An audit event handler manages audit events, sends audit output to a defined location, and controls the output format. IDM provides a number of default audit event handlers, and audit event handlers for third-party log management tools.

Each audit event handler has a set of Common audit event handler properties. Specific audit event handlers have additional configuration properties.

The standard configuration for a new install includes the following handlers:

`JsonAuditEventHandler`

Default state: Enabled

Property: `openidm.audit.handler.json.enabled`

`JsonStdoutAuditEventHandler`

Default state: Disabled

Property: `openidm.audit.handler.stdout.enabled`

`RepositoryAuditEventHandler`

Default state: Disabled

Property: `openidm.audit.handler.repo.enabled`

> **Note**
>
> To change the enable state for any of these handlers, use Property value substitution.

> **Warning**
>
> ForgeRock recommends that you *DO NOT* configure an audit event handler that points to the same repo IDM uses (`RepositoryAuditEventHandler`), as this causes audit records to compete with IDM for resources on the database, which impacts performance.

This command returns the available audit event handlers, along with the audit configuration (in the `conf/audit.json` file):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/audit?_action=availableHandlers"
```

The output includes the configured options for each audit event handler.

To view the audit configuration in the admin UI, click **Configure > System Preferences > Audit**.

The following sections show how to configure the standard audit event handlers. For additional audit event handlers, refer to Audit event handler configuration.

JSON audit event handler

The JSON audit event handler logs events as JSON objects to a set of JSON files. This is the default handler for queries on the audit logs.

> **Note**
>
> Result paging can improve responsiveness when scanning large numbers of audit records through the IDM REST API. The default JSON audit handler does not support paging. If you need to page audit results, use a handler that does support paging, such as the Repository Audit Event Handler.

The following excerpt of an `audit.json` file shows a sample JSON audit event handler configuration:

```
"eventHandlers" : [
    {
        "class" : "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
        "config" : {
            "name" : "json",
            "enabled" : {
                "$bool" : "&{openidm.audit.handler.json.enabled|true}"
            },
            "logDirectory" : "&{idm.data.dir}/audit",
            "buffering" : {
                "maxSize" : 100000,
                "writeInterval" : "100 millis"
            },
            "topics" : [
                "access",
                "activity",
                "sync",
                "authentication",
                "config"
            ]
        }
    },
```

A JSON audit event handler configuration includes the following mandatory properties:

`name`

The audit event handler name (`json`).

`logDirectory`

The name of the directory in which the JSON log files should be written, relative to the *working location*. For more information on the working location, refer to Startup configuration.

You can use property value substitution to direct log files to another location on the filesystem. For more information, refer to Property value substitution.

`buffering - maxSize`

The maximum number of events that can be buffered. The default (and minimum) number of buffered events is 100000.

`buffering - writeInterval`

The delay after which the file-writer thread is scheduled to run after encountering an empty event buffer. The default delay is 100 milliseconds.

`topics`

The list of topics for which audit events are logged.

One JSON file is created for each audit topic that is included in this list:

`access.audit.json`
`activity.audit.json`
`authentication.audit.json`
`config.audit.json`
`sync.audit.json`

> **Note**
>
> Reconciliations are available as an audit topic, but are not enabled by default. To enable auditing on reconciliations, add `recon` to the list of topics. This will add a `recon.audit.json` file to the audit directory.

If you want to get information about a reconciliation without enabling the audit topic, you can get similar details from the `recon/assoc` endpoint. For more information about recon association data, refer to Viewing Reconciliation Association Details.

For a description of all the configurable properties of the JSON audit event handler, refer to JSON Audit Event Handler Properties.

The following excerpt of an `authentication.audit.json` file shows the log message format for authentication events:

```
{
        "context": {
                "ipAddress": "0:0:0:0:0:0:0:1"
        },
        "entries": [{
                "moduleId": "JwtSession",
                "result": "FAILED",
                "reason": {},
                "info": {}
        },
 ...
        {
                "moduleId": "INTERNAL_USER",
                "result": "SUCCESSFUL",
                "info": {
                        "org.forgerock.authentication.principal": "openidm-admin"
                }
        }],
        "principal": ["openidm-admin"],
        "result": "SUCCESSFUL",
        "userId": "openidm-admin",
        "transactionId": "94b9b85f-fbf1-4c4c-8198-ab1ff52ed0c3-24",
        "timestamp": "2016-10-11T12:12:03.115Z",
        "eventName": "authentication",
        "trackingIds": ["5855a363-a1e0-4894-a2dc-fd5270fb99d1"],
        "_id": "94b9b85f-fbf1-4c4c-8198-ab1ff52ed0c3-30"
} {
        "context": {
                "component": "internal/user",
                "roles": ["internal/role/openidm-admin", "internal/role/openidm-authorized"],
                "ipAddress": "0:0:0:0:0:0:0:1",
                "id": "openidm-admin",
                "moduleId": "INTERNAL_USER"
        }...
```

JSON standard output audit event handler

Standard output is also known as `stdout`. A JSON stdout handler sends messages to standard output. The following code is an excerpt of the `audit.json` file, which depicts a sample JSON stdout audit event handler configuration:

```
{
    "class" : "org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHandler",
    "config" : {
        "name" : "stdout",
        "enabled" : {
            "$bool" : "&{openidm.audit.handler.stdout.enabled|false}"
        },
        "topics" : [
            "access",
            "activity",
            "sync",
            "authentication",
            "config"
        ]
    }
}...
```

CSV audit event handler

The CSV audit event handler logs events to a comma-separated value (CSV) file.

> **Important**
>
> The CSV handler does not sanitize messages when writing to CSV log files.
> Do not open CSV logs in spreadsheets and other applications that treat data as code.

The following excerpt of the `audit.json` file shows a sample CSV handler configuration:

```
"eventHandlers" : [
{
    "class" : "org.forgerock.audit.events.handlers.csv.CSVAuditEventHandler",
    "config" : {
        "name" : "csv",
        "logDirectory" : "&{idm.data.dir}/audit",
        "topics" : [ "access", "activity", "sync", "authentication", "config" ]
    }
}
```

The `logDirectory` indicates the name of the directory in which log files should be written, relative to the *working location*. For more information on the working location, refer to Startup configuration.

You can use property value substitution to direct logs to another location on the filesystem. For more information, refer to Property Value Substitution.

If you set up a custom CSV handler, you may configure over 20 different properties, as described in Common Audit Event Handler Properties.

Audit file names are fixed and correspond to the event being audited:

`access.csv`
`activity.csv`
`authentication.csv`
`config.csv`
`recon.csv`
`sync.csv`

Restrictions on configuring the CSV audit handler in the admin UI

If you configure the CSV handler in the admin UI, set at least the following properties:

- The `logDirectory`, the full path to the directory with audit logs, such as `/path/to/openidm/audit`. You can substitute &{idm.install.dir} for `/path/to/openidm`.
- Differing entries for the quote character, `quoteChar` and delimiter character, `delimiterChar`.

  After you have set these options, *do not change them* in the admin UI. Rather, rotate any CSV audit files and edit the configuration properties directly in `conf/audit.json`. Changing the properties in the admin UI generates an error in the console.

- If you enable the CSV tamper-evident configuration, include the `keystoreHandlerName`, *or* a `filename` and `password`. Do not include all three options.

  Before including tamper-evident features in the audit configuration, set up the keys as described in Configure Keys to Protect Audit Logs.

  > **Note**
  >
  > The `signatureInterval` property supports time settings in a human-readable format (default = 1 hour).
  > Examples of allowable `signatureInterval` settings are:
  > - 3 days, 4 m
  > - 1 hour, 3 sec
  >
  > Allowable time units include:
  > - days, day, d
  > - hours, hour, h
  > - minutes, minute, min, m
  > - seconds, second, sec, s

Configure Tamper Protection for CSV Audit Logs

Tamper protection can ensure the integrity of audit logs written to CSV files. You can activate tamper protection in the `audit.json` file directly, or by editing the CSV Audit Event Handler in the admin UI.

Before you change the audit configuration for tamper protection, move or delete any current audit CSV files:

```
mv /path/to/openidm/audit/*.csv /tmp
```

Tamper protection requires keys in the default IDM keystore. If you have not already done so, import a certificate into the keystore, or create your own self-signed certificate:

IDM includes a Java Cryptography Extension Keystore (JCEKS), `keystore.jceks`, in the `/path/to/openidm/security` directory.

Initialize a key pair using the RSA encryption algorithm, using the SHA256 hashing mechanism:

```
keytool \
 -genkeypair \
 -alias "Signature" \
 -dname CN=openidm \
 -keystore /path/to/openidm/security/keystore.jceks \
 -storepass changeit \
 -storetype JCEKS \
 -keypass changeit \
 -keyalg RSA \
 -sigalg SHA256withRSA
```

You can now set up a secret key, in Hash-based message authentication code, using the SHA256 hash function (`HmacSHA256`):

```
keytool \
 -genseckey \
 -alias "Password" \
 -keystore /path/to/openidm/security/keystore.jceks \
 -storepass changeit \
 -storetype JCEKS \
 -keypass changeit \
 -keyalg HmacSHA256 \
 -keysize 256
```

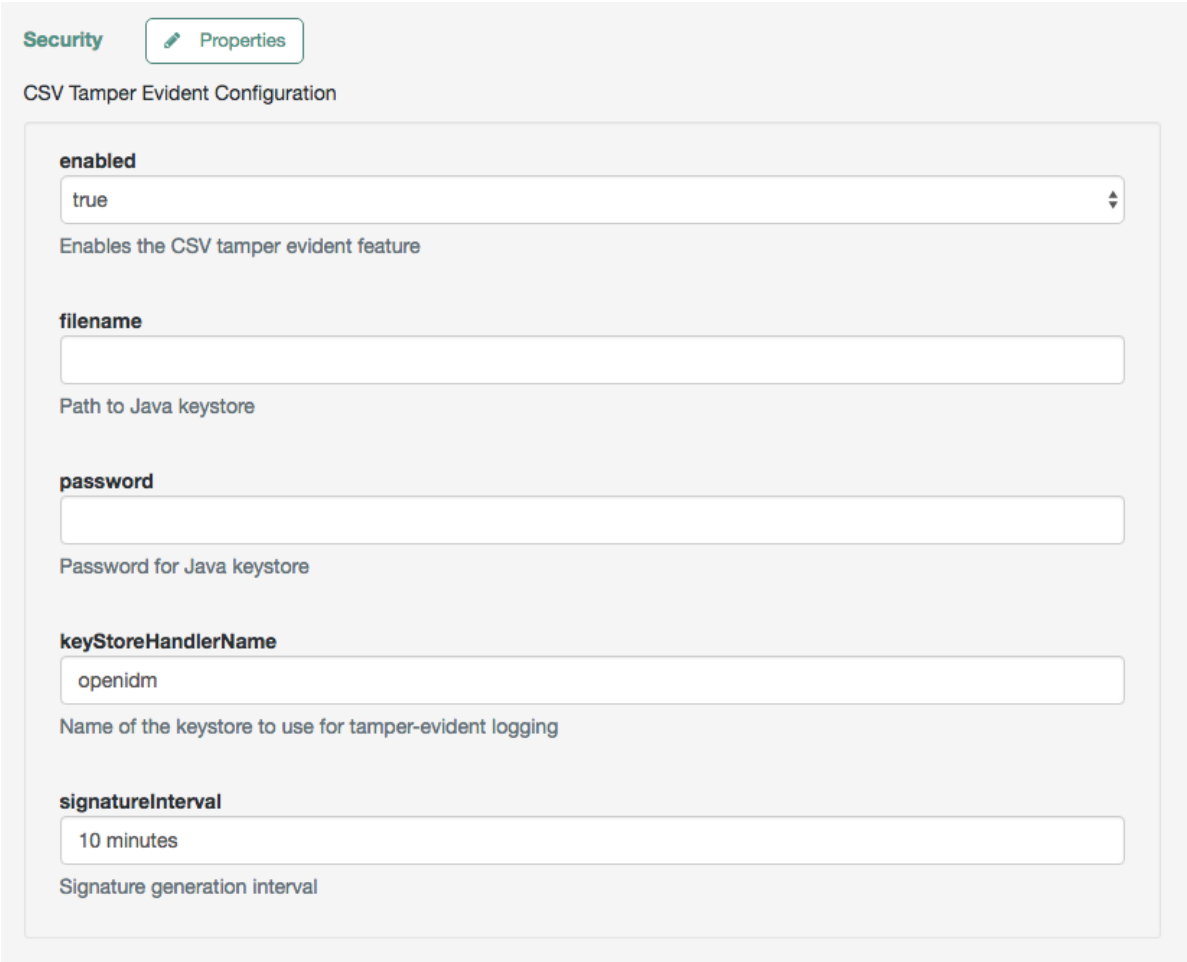
To configure tamper protection, add a `security` property to the CSV audit handler configuration in your `conf/audit.conf` file:

```
{
   "class" : "org.forgerock.audit.handlers.csv.CsvAuditEventHandler",
   "config" : {
   ...
      "security" : {
        "enabled" : true,
        "filename" : "",
        "password" : "",
        "keyStoreHandlerName" : "openidm",
        "signatureInterval" : "10 minutes"
      },
     ...
```

This excerpt shows a tamper-evident configuration where a signature is written to a new line in each CSV file, every 10 minutes. The signature uses the default keystore, configured in the `install-dir/resolver/boot.properties` file. The properties are described in Common audit event handler properties.

To configure tamper protection in the admin UI:

1. Click **Configure > System Preferences > Audit**, and select an existing CSV audit handler, or add a new one.
2. Scroll down to **Security**, and set the keystore options.



When you have saved the configuration changes, you should see the following files in the `/path/to/openidm/audit` directory:

```
tamper-evident-access.csv
tamper-evident-access.csv.keystore
tamper-evident-activity.csv
tamper-evident-activity.csv.keystore
tamper-evident-authentication.csv
tamper-evident-authentication.csv.keystore
tamper-evident-config.csv
tamper-evident-config.csv.keystore
tamper-evident-recon.csv
tamper-evident-recon.csv.keystore
tamper-evident-sync.csv
tamper-evident-sync.csv.keystore
```

When you have configured tamper protection, you can periodically check the integrity of your log files:

The following command verifies audit files in the `--archive` directory (`audit/`), that belong to the access `--topic` , verified with the `keystore.jceks` keystore, using the CSV audit handler bundle, `forgerock-audit-handler-csv-version.jar` :

```
java -jar \
bundle/forgerock-audit-handler-csv-version.jar \
--archive audit/ \
--topic access \
--keystore security/keystore.jceks \
--password changeit
```

If there are changes to your `tamper-evident-access.csv` file, a message similar to the following displays:

```
FAIL tamper-evident-access.csv-2016.05.10-11.05.43 The HMac at row 3 is not correct.
```

Note

Note the following restrictions on verifying CSV audit files:

- You can only verify audit files that have already been rotated. You cannot verify an audit file that is currently being written to.
- Verification of tampering is supported only for CSV audit files with the following format:

```
"formatting" : {
    "quoteChar" : "\"",
    "delimiterChar" : ",",
    "endOfLineSymbols" : "\n"
},
```

- A tamper-evident audit configuration rotates files automatically and pairs the rotated file with the required keystore file. Files that are rotated manually cannot be verified, as the required keystore information is not appended.

Router Audit Event Handler

The router audit event handler logs events to any external or custom endpoint, such as `system/scriptedsql` or `custom-endpoint/myhandler`. It uses target-assigned values of `_id`.

A sample configuration for a `router` event handler is provided in the `audit.json` file in the `openidm/samples/audit-jdbc/conf` directory, and described in About the Configuration Files. This sample directs log output to a JDBC repository. The audit configuration file (`conf/audit.json`) for the sample shows the following event handler configuration:

```
{
    "class": "org.forgerock.openidm.audit.impl.RouterAuditEventHandler",
    "config": {
        "name": "router",
        "topics" : [ "access", "activity", "sync", "authentication", "config" ],
        "resourcePath" : "system/auditdb"
    }
},
```

The `resourcePath` property in the configuration indicates that logs should be directed to the `system/auditdb` endpoint. This endpoint, and the JDBC connection properties, are defined in the connector configuration file (`conf/provisioner.openicf-auditdb.json`), as follows:

```
{
    "configurationProperties" : {
        "username" : "root",
        "password" : "password",
        "driverClassName" : "com.mysql.cj.jdbc.Driver",
        "url" : "jdbc:mysql://&{openidm.repo.host}:&{openidm.repo.port}/audit",
        "autoCommit" : true,
        "jdbcDriver" : "com.mysql.cj.jdbc.Driver",
        "scriptRoots" : ["&{idm.instance.dir}/tools"],
        "createScriptFileName" : "CreateScript.groovy",
        "testScriptFileName" : "TestScript.groovy",
        "searchScriptFileName" : "SearchScript.groovy"
    },
...
```

Include the correct URL or IP address of your remote JDBC repository in the `boot.properties` file for your project.

When JSON information is sent to the router audit event handler, the value of `_id` is replaced with `eventId` .

Repository Audit Event Handler

The repository audit event handler sends information to a JDBC repository. If you are using ForgeRock Directory Services (DS) as the repository, you cannot enable this audit event handler, because audit data cannot be stored in DS.

> **Warning**
>
> ForgeRock recommends that you *DO NOT* use the `RepositoryAuditEventHandler` , as this causes audit records to compete with IDM for resources on the database, which impacts performance.

Log entries are stored in the following tables of a JDBC repository:

- `auditaccess`
- `auditactivity`
- `auditauthentication`
- `auditconfig`
- `auditrecon`
- `auditsync`

You can use the repository audit event handler to generate reports that combine information from multiple tables.

Each of these JDBC tables maps to an object in the database table configuration file (`repo.jdbc.json`). The following excerpt of that file illustrates the mappings for the `auditauthentication` table:

```
"audit/authentication" : {
    "table" : "auditauthentication",
    "objectToColumn" : {
        "_id" : "objectid",
        "transactionId" : "transactionid",
        "timestamp" : "activitydate",
        "userId" : "userid",
        "eventName" : "eventname",
        "result" : "result",
        "principal" : {"column" : "principals", "type" : "JSON_LIST"},
        "context" : {"column" : "context", "type" : "JSON_MAP"},
        "entries" : {"column" : "entries", "type" : "JSON_LIST"},
        "trackingIds" : {"column" : "trackingids", "type" : "JSON_LIST"},
    }
},
```

The tables correspond to the `topics` listed in the `audit.json` file. For example:

```
{
    "class": "org.forgerock.openidm.audit.impl.RepositoryAuditEventHandler",
    "config": {
        "name": "repo",
        "topics" : [ "access", "activity", "sync", "authentication", "config" ]
    }
},
```

JMS audit event handler

The Java Message Service (JMS) is a Java API for sending asynchronous messages between clients. IDM audit information can be handled by the JMS audit event handler, which sends information to message brokers. The message brokers can then forward that information to external log analysis systems.

The JMS audit event handler works with the following message brokers:

- Apache ActiveMQ Artemis.

  For a demonstration, refer to Direct audit information to a JMS broker.

- TIBCO Enterprise Message Service, as described in this chapter.

This implementation supports the *publish/subscribe* model. For more information, refer to Basic JMS API Concepts.

> **Important**
>
> The JMS audit event handler does not support queries. If you enable JMS, and need to query audit events, you must enable a second audit handler that supports queries. Specify that audit handler in the `audit.json` file with the `handlerForQueries` property, or in the admin UI with the **Use For Queries** option.

The JMS audit event handler supports JMS communication, based on the following components:

- A JMS message broker that provides clients with connectivity, along with message storage and message delivery functionality.
- JMS messages that follow a specific format, described in JMS Message Format.

- Destinations external to IDM and the message broker. IDM (including the audit service) is a *producer* and not a destination. IDM sends messages to a topic in a message broker. Consumers (clients) subscribe to the message broker.

> **Note**
>
> *JMS Topics* are not the same as the ForgeRock audit event `topics` listed in your project's `audit.json` file. For more information about JMS topics, refer to the documentation on the publish/subscribe model. ForgeRock audit event topics specify categories of events (including access, activity, authentication, configuration, reconciliation, and synchronization). These event topics are published via the audit handler(s).

Dependencies for JMS messaging

The JMS audit event handler requires Apache ActiveMQ Artemis and additional dependencies bundled with the ActiveMQ Artemis delivery. This section lists the dependencies, and where they must be installed in the IDM instance. If you use a different ActiveMQ version, you may need to download the corresponding dependencies separately.

1. Download the following files:
   - Apache ActiveMQ Artemis.

     > **Note**
     >
     > This sample was tested with version 2.20.0.

   - The most recent `bnd` JAR file from https://repo1.maven.org/maven2/biz/aQute/bnd/biz.aQute.bnd/.

     > **Tip**
     >
     > The bnd utility lets you create OSGi bundles for libraries that do not support OSGi.

2. Unpack the ActiveMQ Artemis archive. For example:

```
tar -zxvf ~/Downloads/apache-artemis-2.20.0-bin.tar.gz
```

3. Create a temporary directory, and then change to that directory:

```
mkdir ~/Downloads/tmp
cd ~/Downloads/tmp/
```

4. Move the ActiveMQ Artemis Client and `bnd` JAR files to the temporary directory.

```
mv ~/Downloads/apache-artemis-2.20.0/lib/client/artemis-jms-client-all-2.20.0.jar ~/Downloads/tmp/
mv ~/Downloads/biz.aQute.bnd-version.jar ~/Downloads/tmp/
```

5. Create an OSGi bundle:
   1. In a text editor, create a BND file named `activemq.bnd` with the following contents, and save it to the current directory:

```
version=2.20.0
Export-Package: *;version=${version}
Import-Package: !org.apache.log4j.*,!org.apache.log.*,!org.apache.avalon.framework.logger.*,!
org.apache.avalon.framework.logger.*,!org.glassfish.json.*,!org.conscrypt.*,!
org.apache.logging.*,!org.bouncycastle.jsse.*,!org.eclipse.*,!sun.security.*,!reactor.*,!
org.apache.activemq.artemis.shaded.*,!com.aayushatharva.*,!com.github.luben.zstd,!
com.jcraft.jzlib,!com.ning.compress,!com.ning.compress.lzf,!com.ning.compress.lzf.util,!
com.oracle.svm.core.annotate,!lzma.*,!net.jpountz.*,*
Bundle-Name: ActiveMQArtemis :: Client
Bundle-SymbolicName: org.apache.activemq
Bundle-Version: ${version}
```

Your `tmp/` directory should now contain the following files:

```
ls -1 ~/Downloads/tmp/
activemq.bnd
artemis-jms-client-all-2.20.0.jar
biz.aQute.bnd-version.jar
```

2. In the same directory, create the OSGi bundle archive file. For example:

```
java -jar biz.aQute.bnd-version.jar wrap \
--properties activemq.bnd \
--output artemis-jms-client-all-2.20.0-osgi.jar \
artemis-jms-client-all-2.20.0.jar
```

6. Copy the resulting `artemis-jms-client-all-2.20.0-osgi.jar` file to the `openidm/bundle` directory:

```
cp artemis-jms-client-all-2.20.0-osgi.jar /path/to/openidm/bundle/
```

Configure the JMS audit event handler

You can configure the JMS audit event handler in the admin UI, or in your `conf/audit.json` file.

To configure the JMS audit event handler in the admin UI:

1. Select **Configure > System Preferences > Audit**.
2. Under **Event Handlers**, select **JmsAuditEventHandler > Add Event Handler**.

The event handler configuration properties are discussed in this section. For a complete list of configuration options, refer to JMS Audit Event Handler Properties.

To configure the audit event handler in the `conf/audit.json` file, refer to the sample configuration provided in `/path/to/openidm/samples/audit-jms/conf/audit.json`. The following excerpt of that file shows the JMS audit event handler configuration:

```
{
    "class" : "org.forgerock.audit.handlers.jms.JmsAuditEventHandler",
    "config" : {
        "name": "jms",
        "enabled" : true,
        "topics": [
            "access",
            "activity",
            "config",
            "authentication",
            "sync",
            "recon"
        ],
        "deliveryMode": "NON_PERSISTENT",
        "sessionMode": "AUTO",
        "batch": {
            "writeInterval": "1 second",
            "capacity": 1000,
            "maxBatchedEvents": 100
        },
        "jndi": {
            "contextProperties": {
                "java.naming.factory.initial" : "org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory",
                "java.naming.provider.url" : "tcp://127.0.0.1:61616?daemon=true",
                "topic.forgerock.idm.audit" : "forgerock.idm.audit"
            },
            "topicName": "forgerock.idm.audit",
            "connectionFactoryName": "ConnectionFactory"
        }
    }
}
```

In this sample configuration, the JMS audit event handler is `enabled`, with `NON_PERSISTENT` delivery of audit events in batches. The handler is configured to use the Apache ActiveMQ Artemis Java Naming and Directory Interface (JNDI) message broker, on port 61616.

For an example of how to configure Apache ActiveMQ Artemis, refer to Direct audit information to a JMS broker.

If you substitute a different JNDI message broker, change the `jndi.contextProperties` accordingly. If you configure the JNDI message broker on a remote system, substitute the corresponding IP address.

Configure SSL for Apache ActiveMQ Artemis

For information on configuring Apache ActiveMQ Artemis security features, including SSL, refer to the ActiveMQ Artemis Documentation:

- Security
- Configuring the Transport

JMS message format

The following JMS message reflects the authentication of the `openidm-admin` user, logging into the admin UI from a remote location, IP address 172.16.209.49.


```
{
  "event": {
    "_id": "134ee773-c081-436b-ae61-a41e8158c712-565",
    "trackingIds": [
      "4dd1f9de-69ac-4721-b01e-666df388fb17",
      "185b9120-406e-47fe-ba8f-e95fd5e0abd8"
    ],
  "context": {
    "id": "openidm-admin",
    "ipAddress": "172.16.209.49",
    "roles": [
      "internal/role/openidm-admin",
      "internal/role/openidm-authorized"
    ],
    "component": "internal/user"
  },
  "entries": [
    {
      "info": {
        "org.forgerock.authentication.principal": "openidm-admin"
      },
      "result": "SUCCESSFUL",
      "moduleId": "JwtSession"
    }
  ],
  "principal": [
    "openidm-admin"
  ],
    "result": "SUCCESSFUL",
    "userId": "openidm-admin",
    "transactionId": "134ee773-c081-436b-ae61-a41e8158c712-562",
    "timestamp": "2016-04-15T14:57:53.114Z",
    "eventName": "authentication"
  },
  "auditTopic": "authentication"
}
```


JMS, TIBCO, and SSL

You can integrate the JMS audit event handler with the TIBCO Enterprise Message Service.

You'll need to use two bundles from your TIBCO installation: `tibjms.jar`, and if you're setting up a secure connection, `tibcrypt.jar`. With the following procedure, you'll process `tibjms.jar` into an OSGi bundle:

1. Download the most recent `bnd` JAR file from https://repo1.maven.org/maven2/biz/aQute/bnd/biz.aQute.bnd/. The bnd utility lets you create OSGi bundles for libraries that do not yet support OSGi. If you have previously set up the ActiveMQ Artemis server, you may have already downloaded this file.

2. In the same directory, create a file named `tibco.bnd`, and add the following lines to that file:

```
version=8.3.0
Export-Package: *;version=${version}
Bundle-Name: TIBCO Enterprise Message Service
Bundle-SymbolicName: com/tibco/tibjms
Bundle-Version: ${version}
```

3. Add the `tibco.jar` file to the same directory.
4. Run the following command to create the bundle:

```
java \
-jar biz.aQute.bnd-version.jar wrap \
-properties tibco.bnd tibjms.jar
```

5. Rename the newly created `tibjms.bar` file to `tibjms-osgi.jar` , and copy it to the `/path/to/openidm/bundle` directory.
6. If you're configuring SSL, copy the `tibcrypt.jar` file from your TIBCO installation to the `/path/to/openidm/bundle` directory.

You also need to configure your project's `audit.conf` configuration file. The options are similar to those listed earlier in Configure the JMS Audit Event Handler, except for the following `jndi` code block:

```
"jndi": {
   "contextProperties": {
      "java.naming.factory.initial" : "com.tibco.tibjms.naming.TibjmsInitialContextFactory",
      "java.naming.provider.url" : "tibjmsnaming://localhost:7222"
   },
   "topicName": "audit",
   "connectionFactoryName": "ConnectionFactory"
}
```

If your TIBCO server is on a remote system, substitute appropriately for `localhost` . If you're configuring a secure TIBCO installation, you'll want to configure a different code block:

```
"jndi": {
   "contextProperties": {
      "java.naming.factory.initial" : "com.tibco.tibjms.naming.TibjmsInitialContextFactory",
      "java.naming.provider.url" : "ssl://localhost:7243",
      "com.tibco.tibjms.naming.security_protocol" : "ssl",
      "com.tibco.tibjms.naming.ssl_trusted_certs" : "/path/to/tibco/server/certificate/cert.pem",
      "com.tibco.tibjms.naming.ssl_enable_verify_hostname" : "false"
   },
   "topicName": "audit",
   "connectionFactoryName": "SSLConnectionFactory"
}
```

Do not add the TIBCO certificate to the IDM `truststore` . The formats are not compatible.

When this configuration work is complete, don't forget to start your TIBCO server before starting IDM. For more information, refer to the TIBCO Enterprise Message Service Users's Guide.

Syslog audit event handler

The Syslog audit event handler lets you log messages to a Syslog server, based on the Syslog Protocol.

You can configure the Syslog audit event handler in the admin UI, or in your project's `conf/audit.json` file. The following excerpt from this file shows a possible Syslog configuration:

```
{
    "class" : "org.forgerock.audit.handlers.syslog.SyslogAuditEventHandler",
    "config" : {
        "protocol" : "UDP",
        "host" : "172.16.206.5",
        "port" : 514,
        "connectTimeout" : 5,
        "facility" : "KERN",
        "severityFieldMappings" : [
            {
                "topic" : "recon",
                "field" : "exception",
                "valueMappings" : {
                    "SEVERE" : "EMERGENCY",
                    "INFO" : "INFORMATIONAL"
                }
            }
        ],
        "buffering" : {
            "enabled" : false
        },
        "name" : "syslog1",
        "topics" : [
            "config",
            "activity",
            "authentication",
            "access",
            "recon",
            "sync"
        ],
        "enabled" : true
    }
}
```

The `name`, `topics`, and `enabled` options in the last part of the excerpt are common to all audit event handlers. For detailed information on the remaining properties, refer to Syslog Audit Event Handler Properties.

Audit event topics

The audit service logs information from six event topics: access, activity, authentication, configuration, reconciliation, and synchronization.

When you start IDM, it creates audit log files in the `openidm/audit` directory. The default file-based audit event handler is the JSON handler, which creates one JSON file for each event topic.

To configure default and custom audit topics in the admin UI, select **Configure > System Preferences**. Click on the **Audit** tab, and scroll down to **Event Topics**.

Default audit event topics

The audit service logs the following event topics by default:

Access Events

IDM writes messages at *system boundaries*, that is REST endpoints and the invocation of scheduled tasks in this log. In short, it includes who, what, and output for every access request.

Default file: `openidm/audit/access.audit.json`

Activity Events

IDM logs operations on internal (managed) and external (system) objects to this log.

Entries in the activity log contain identifiers, both for the action that triggered the activity, and for the original caller and the relationships between related actions, on internal and external objects.

Default file: `openidm/audit/activity.audit.json`

Authentication Events

IDM logs the results of authentication operations to this log, including situations and the actions taken on each object, including when and how a user authenticated and related events. The activity log contains additional detail about each authentication action.

Default file: `openidm/audit/authentication.audit.json`

Configuration Events

IDM logs the changes to the configuration in this log. The configuration log includes the "before" and "after" settings for each configuration item, with timestamps.

Default file: `openidm/audit/config.audit.json`

Reconciliation Events

IDM logs the results of reconciliation runs to this log (including situations and the resulting actions taken). The activity log contains details about the actions, where log entries display parent activity identifiers, `recon/reconID`, links, and policy events by data store.

Default file: `openidm/audit/recon.audit.json`

Synchronization Events

IDM logs the results of automatic synchronization operations (liveSync and implicit synchronization) to this log, including situations and the actions taken on each object, by account. The activity log contains additional detail about each action.

Default file: `openidm/audit/sync.audit.json`

For detailed information about each audit event topic, refer to Audit event handler configuration.

Custom audit event topics

You can create custom event topics to collect audit information for customizations, such as scripts. Creating a new event topic has a few additional requirements:

- You must specify a schema for your custom topic. The schema determines the structure and type of information stored in audit logs.
- Your script needs to call the new audit event topic (for example `audit/example`), providing the values you specified in your topic schema.

Create custom event topics directly in `audit.json`, or using the admin UI. The following example, from an `audit.json` file, has been modified to include a custom audit event topic named `example`:

```
"eventTopics": {
  "authentication": {},
  "access": {},
  ...
  "example": {
    "schema": {
      "$schema": "http://json-schema.org/draft-04/schema#",
      "id": "/",
      "type": "object",
      "properties": {
        "_id": {
          "id": "_id",
          "type": "string"
        },
        "transactionId": {
          "id": "transactionId",
          "type": "string"
        },
        "timestamp": {
          "id": "timestamp",
          "type": "string"
        },
        "status": {
          "id": "status",
          "type": "string"
        },
        "message": {
          "id": "message",
          "type": "string"
        }
      },
      "filter": {
        "actions": []
      }
    }
  }
}
```

When your topic has been created, add it to an event handler such as the `JsonAuditEventHandler`, in order to output the audit logs in your desired format. New audit events can be sent by calling the audit topic endpoint (in this example, `audit/example`). For example, the following REST call will add a new audit event for the `example` topic:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
   "transactionId": "779d3cda-dab3-4e54-9ab1-e0ca4c7ae6df-699",
   "timestamp": "2019-02-12T01:11:02.675Z",
   "status": "SUCCESS",
   "message": "Script has run successfully."
}' \
"http://localhost:8080/openidm/audit/example"
{
  "_id": "2091c3f2-7a22-47bf-a618-b2af4c322e46-1192",
  "transactionId": "779d3cda-dab3-4e54-9ab1-e0ca4c7ae6df-699",
  "timestamp": "2019-02-12T01:11:02.675Z",
  "status": "SUCCESS",
  "message": "Script has run successfully."
}
```

This new audit event will be logged to the audit log specified by your event handler. For example, if you had added the `example` topic to the `JsonAuditEventHandler`, you can find your new audit event logged in `audit/example.audit.json`.

Filter audit data

The audit configuration (in `conf/audit.json`) includes a `filter` parameter that lets you specify what should be logged, per event topic. The information that is logged can be filtered in various ways.

The following excerpt of a sample `audit.json` file shows the filter element for the activity log:

```
"eventTopics" : {
    "authentication" : { },
    "access" : { },
    "activity" : {
        "filter" : {
            "actions" : [
                "create",
                "update",
                "delete",
                "patch",
                "action"
            ]
        },
  ...
}
```

To configure audit filtering in the admin UI, select **Configure > System Preferences > Audit**. Scroll down to **Event Topics**, and click the pencil icon next to the event that you want to filter. The filter tabs, **Filter Actions**, **Filter Fields**, **Filter Script**, and **Filter Triggers**, correspond to the filtering capabilities discussed here.

Filter by action

The `filter` `actions` list enables you to specify the actions that are logged, per event type. This filter is essentially a `fields` filter (as described in Filter by Field Value) that filters log entries by the value of their `actions` field.

The following configuration specifies that the actions create, update, delete, patch, and action should be included in the log, for the activity audit event topic.

```
"eventTopics" : {
...
    "activity": {
        "filter" : {
            "actions" : [
                "create",
                "update",
                "delete",
                "patch",
                "action"
            ]
        },
        ...
    }
}
```

The list of actions that can be filtered into the log depend on the event type. The following table lists the actions that can be filtered, per event type.

Actions that can be Filtered Per Event Type

Event Type	Actions	Description
Activity and Configuration	`read`	When an object is read by using its identifier. By default, read actions are not logged. Note that due to the potential result size in the case of read operations on `system/` endpoints, only the read is logged, and not the resource detail. If you really need to log the complete resource detail, set the following property in your `resolver/boot.properties` file: `openidm.audit.logFullObjects=true`
	`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. (Activity only.)

Event Type	Actions	Description
	`query`	When a query is performed on an object. By default, query actions are not logged. Note that, due to the potential result size in the case of query operations on `system/` endpoints, only the query is logged, and not the resource detail. If you really need to log the complete resource detail, add the following line to your `resolver/boot.properties` file: `openidm.audit.logFullObjects=true`
	`action`	When an action is performed on an object. (Activity only.)
Reconciliation and Synchronization	`create`	When a target object is created.
	`delete`	When a target object is deleted.
	`update`	When a target object is updated.
	`link`	When a link is created between a source object and an existing target object.
	`unlink`	When a link is removed between a source object and a target object.
	`exception`	When the synchronization situation results in an exception. For more information, refer to Synchronization situations and actions.
	`ignore`	When the target object is ignored; that is, no action is taken.
Authentication and Access	`-`	No actions can be specified for the authentication or the access log event type.

Filter by field value

You can add a list of `filter` `fields` to the audit configuration, that lets you filter log entries by specific fields. For example, you might want to restrict the reconciliation or audit log so that only summary information is logged for each reconciliation operation. The following addition to the `audit.json` file specifies that entries are logged in the reconciliation log only if their `entryType` is `start` or `summary`.

```
"eventTopics" : {
    ...
    "activity" : {
        "filter" : {
            "actions" : [
                "create",
                "update",
                "delete",
                "patch",
                "action
            ],
            "fields" : [
                {
                    "name" : "entryType",
                    "values" : [
                        "start",
                        "summary"
                    ]
                }
            ]
        }
    }
    ...
},
...
```

To use nested properties, specify the field name as a JSON pointer. For example, to filter entries according to the value of the `authentication.id`, you would specify the field name as `authentication/id`.

Filter with a script

Apart from the audit filtering options described in the previous sections, you can use a JavaScript or Groovy script to filter what is logged. Audit filter scripts are referenced in the audit configuration file (`conf/audit.json`), and can be configured per event type. The following sample configuration references a script named `auditfilter.js`, which is used to limit what is logged in the reconciliation audit log:

```
{
    "eventTopics" : {
        ...
        "recon" : {
            "filter" : {
                "script" : {
                    "type" : "text/javascript",
                    "file" : "auditfilter.js"
                }
            }
        },
        ...
}
```

The `request` and `context` objects are available to the script. Before writing the audit entry, IDM can access the entry as a `request.content` object. For example, to set up a script to log just the summary entries for mapping managed users in an LDAP data store, you could include the following in the `auditfilter.js` script:

```
(function() {
    return request.content.entryType == 'summary' &&
    request.content.mapping == 'systemLdapAccounts_managedUser'
}());
```

The script must return `true` to include the log entry; `false` to exclude it.

Filter by trigger

You can add a `filter` `triggers` list to the audit configuration, that specifies the actions that will be logged for a specific trigger. For example, the following addition to the `audit.json` file specifies that only `create` and `update` actions are logged for in the activity log, for an activity that was triggered by a `recon`.

```
"eventTopics" : {
    "activity" : {
        "filter" : {
            "actions" : [
            ...
            ],
            "triggers" : {
                "recon" : [
                    "create",
                    "update"
                ]
            }
    ...
```

If a trigger is provided, but no actions are specified, nothing is logged for that trigger. If a trigger is omitted, all actions are logged for that trigger. Only the `recon` trigger is implemented. For a list of reconciliation actions that can be logged, refer to Synchronization Actions.

Use policies to filter audit data

In addition to event-based filtering, you can use policies to include or exclude specific information in the audit logs. By default, IDM safelists fields that are safe to log. To include or exclude additional fields or values, edit `conf/audit.json`:

```
"filterPolicies" : {
    "value" : {
        "excludeIf" : [ ],
        "includeIf" : [ ]
    }
}
```

> **Note**
>
> Although you can't edit the default safelist, IDM processes the safelist before the blocklist, so any items added to `excludeIf` override their safelist status.

- To specify data to exclude from audit logs, use the `excludeIf` property.
  - To exclude an entire field, use the `field` property.
  - To exclude a field that contains a specific value, use the `value` property.
- To specify data to include in *custom* audit event logs, use the `includeIf` property.

> **Note**
>
> This setting has no effect on default audit event topics.

Default audit log safelists by event topic

- /_id
- /timestamp
- /eventName
- /transactionId
- /trackingIds
- /userId
- /client
- /server
- /http/request/secure
- /http/request/method
- /http/request/path
- /http/request/headers/accept
- /http/request/headers/accept-api-version
- /http/request/headers/content-type
- /http/request/headers/host
- /http/request/headers/user-agent
- /http/request/headers/x-forwarded-for
- /http/request/headers/x-forwarded-host
- /http/request/headers/x-forwarded-port
- /http/request/headers/x-forwarded-proto

- /http/request/headers/x-original-uri
- /http/request/headers/x-real-ip
- /http/request/headers/x-request-id
- /http/request/headers/x-requested-with
- /http/request/headers/x-scheme
- /request
- /response
- /roles
- /_id
- /timestamp
- /eventName
- /transactionId
- /trackingIds
- /userId
- /runAs
- /objectId
- /operation
- /changedFields
- /revision
- /status
- /message
- /passwordChanged
- /context
- /provider
- /_id
- /timestamp
- /eventName
- /transactionId
- /trackingIds
- /userId

- /principal
- /entries
- /result
- /provider
- /method
- /_id
- /timestamp
- /eventName
- /transactionId
- /trackingIds
- /userId
- /runAs
- /objectId
- /operation
- /changedFields
- /revision
- /_id
- /action
- /ambiguousTargetObjectIds
- /entryType
- /eventName
- /exception
- /linkQualifier
- /mapping
- /message
- /messageDetail
- /reconAction
- /reconciling
- /reconId
- /situation

- /sourceObjectId
- /status
- /targetObjectId
- /timestamp
- /trackingIds
- /transactionId
- /userId
- /_id
- /action
- /eventName
- /exception
- /linkQualifier
- /mapping
- /message
- /messageDetail
- /situation
- /sourceObjectId
- /status
- /targetObjectId
- /timestamp
- /trackingIds
- /transactionId
- /userId

Configure audit filter policies in the admin UI

1. From the navigation bar, click **Configure > System Preferences**.
2. On the **System Preferences** page, click the **Audit** tab.

   The **Audit Filter Policy** area displays the policies that exist in `conf/audit.json`.
3. Make changes in the **Audit Filter Policy** area, and click **Save**.

Audit filter example

A typical use case for filtering audit data by policy is to keep personally identifiable information (PII) out of the logs. To exclude a specific field from the audit logs, add the field to the `filterPolicies` element, as follows:

```
"filterPolicies" : {
    "value" : {...}
    "field" : {
        "excludeIf" : [
            "/eventTopic/objectURI"
        ]
    }
}
```

Consider the following entry in a sample activity log, showing a change to the `telephoneNumber` field for a user:

```
{
  "_id": "334ed888-3179-4990-b475-c1982403f063-27593",
  "timestamp": "2021-11-09T23:33:25.802Z",
  "eventName": "activity",
  "transactionId": "334ed888-3179-4990-b475-c1982403f063-27554",
  "userId": "openidm-admin",
  "runAs": "openidm-admin",
  "objectId": "managed/user/ba46c2cc-e897-4a69-bb3c-a0c83d9f88bb",
  "operation": "PATCH",
  "changedFields": [],
  "revision": "d4907846-7a84-4da6-898c-a8c9b6f992c5-1210",
  "status": "SUCCESS",
  "message": "",
  "passwordChanged": false
}
```

Because the default Activity Safelist doesn't contain `telephoneNumber`, the change isn't reflected in the audit log.

To include the before and after telephone number in the activity audit log, add the following filter policy to `conf/audit.json`:

```
"filterPolicies" : {
  "field" : {
    "excludeIf" : [ ],
    "includeIf" : [
      "/activity/before/telephoneNumber",
      "/activity/after/telephoneNumber" ]
}
```

With this configuration, a similar change would appear in the activity log as:

```
{
  "before": {
    "telephoneNumber": "360-555-5566"
  },
  "after": {
    "telephoneNumber": "360-555-5555"
  },
  "_id": "334ed888-3179-4990-b475-c1982403f063-28385",
  "timestamp": "2021-11-09T23:35:51.718Z",
  "eventName": "activity",
  "transactionId": "334ed888-3179-4990-b475-c1982403f063-28346",
  "userId": "openidm-admin",
  "runAs": "openidm-admin",
  "objectId": "managed/user/ba46c2cc-e897-4a69-bb3c-a0c83d9f88bb",
  "operation": "PATCH",
  "changedFields": [],
  "revision": "d4907846-7a84-4da6-898c-a8c9b6f992c5-1242",
  "status": "SUCCESS",
  "message": "",
  "passwordChanged": false
}
```


Note

By default, the `/access/http/request/headers` and `/access/http/response/headers` fields are considered case-insensitive for filtering. All other fields are considered case-sensitive.
To specify that a value should be filtered, regardless of case, add the `caseInsensitiveFields` property to your audit configuration, including an array of fields that should be considered case-insensitive. Fields are referenced using JSON pointer syntax and the array of fields can be empty.
With the following configuration, the audit service excludes cookies named `session-jwt` and `session-JWT` from the log:

```
"caseInsensitiveFields" : [
    "http.request.cookies"
],
```

Monitor specific activity log changes

For the activity log only, you can specify fields whose values are considered particularly important in terms of logging.

Fields to watch

The `watchedFields` property (in `conf/audit.json`) lets you define a list of properties that should be monitored for changes. When the value of one of the properties in this list changes, the change is logged in the activity log, under the column `changedFields`. This parameter enables you to have quick access to important changes in the log.

Properties to monitor are listed as values of the `watchedFields` property, separated by commas, for example:

```
"watchedFields" : [ "email", "address" ]
```

You can monitor changes to any field in this way.

To configure watched fields in the admin UI, select **Configure > System Preferences > Audit**. Scroll down to **Event Topics**, and click the pencil icon next to the **activity** event.

Password fields to watch

You can set a list of `passwordFields` that functions much like the `watchedFields` property. Changes to these property values are logged in the activity log, under the column `changedFields`. In addition, when a password property is changed, the boolean `passwordChanged` flag is set to `true` in the activity log. Properties that should be considered as passwords are listed as values of the `passwordFields` parameter, separated by commas. For example:

```
"passwordFields" : [ "password", "userPassword" ]
```

To configure password fields in the admin UI, select **Configure > System Preferences > Audit**. Scroll down to **Event Topics**, and click the pencil icon next to the `activity` event.

Configure an audit exception formatter

The audit service includes an *exception formatter*, configured in the following snippet of the `audit.json` file:

```
"exceptionFormatter" : {
   "type" : "text/javascript",
   "file" : "bin/defaults/script/audit/stacktraceFormatter.js"
},
```

As shown, you may find the script that defines how the exception formatter works in the `stacktraceFormatter.js` file. That file handles the formatting and display of exceptions written to the audit logger.

Change audit write behavior

You can buffer audit logging to minimize the writes on your systems. Configure buffering either in `conf/audit.json`, or using the admin UI.

To configure buffering for specific event handler in the admin UI, click **Configure > System Preferences** and click on the **Audit** tab. When you customize or create an event handler, you can configure the following settings:

Audit Buffering Options

Property	UI Text	Description
`enabled`	True or false	Enables / disables buffering.
`autoFlush`		True or false; whether the Audit Service automatically flushes events after writing them to disk.

The following sample code illustrates where you would configure these properties in the `audit.json` file.

```
...
    "eventHandlers" : [
      {
        "config" : {
          ...
          "buffering" : {
            "autoFlush" : false,
            "enabled" : false
          }
        },
...
```

You can set up `autoFlush` when buffering is enabled. IDM then writes data to audit logs asynchronously, while `autoFlush` functionality ensures that the audit service writes data to logs on a regular basis.

If audit data is important, do activate `autoFlush`. It minimizes the risk of data loss in case of a server crash.

Purge obsolete audit information

If reconciliation audit volumes grow "excessively" large, any subsequent reconciliations, as well as queries to audit tables, can become "sluggish". In a deployment with limited resources, a lack of disk space can affect system performance.

You might already have restricted what is logged in your audit logs by setting up filters, as described in Filter Audit Data. You can also use specific queries to purge reconciliation audit logs, or you can purge reconciliation audit entries older than a specific date, using timestamps.

IDM provides a sample purge script, `autoPurgeRecon.js`, in the `bin/defaults/script/audit` directory. This script purges reconciliation audit log entries only from the internal repository. It does not purge data from the corresponding JSON files or external repositories.

To purge reconciliation audit logs on a regular basis, set up a schedule. A sample schedule is provided in `openidm/samples/example-configurations/schedules/schedule-autoPurgeAuditRecon.json`. You can change that schedule as required, and copy the file to the `conf/` directory of your project, in order for it to take effect.

The sample purge schedule file is as follows:

```
{
   "enabled" : false,
   "type" : "cron",
   "schedule" : "0 0 */12 * * ?",
   "persisted" : true,
   "misfirePolicy" : "doNothing",
   "invokeService" : "script",
   "invokeContext" : {
      "script" : {
         "type" : "text/javascript",
         "file" : "audit/autoPurgeAuditRecon.js",
         "input" : {
            "mappings" : [ "%" ],
            "purgeType" : "purgeByNumOfReconsToKeep",
            "numOfRecons" : 1,
            "intervalUnit" : "minutes",
            "intervalValue" : 1
         }
      }
   }
}
```

For information about the schedule-related properties in this file, refer to Schedule synchronization.

Beyond scheduling, the following parameters are of interest for purging the reconciliation audit logs:

input

Input information. The parameters below specify different kinds of input.

mappings

An array of mappings to prune. Each element in the array can be either a string or an object.

Strings must contain the mapping(s) name and can use "%" as a wild card value that will be used in a LIKE condition.

Objects provide the ability to specify mapping(s) to include/exclude and must be of the form:

```
{
      "include" : "mapping1",
      "exclude" : "mapping2"
      ...
}
```

purgeType

The type of purge to perform. Can be set to one of the following values:

purgeByNumOfReconsToKeep

Uses the `deleteFromAuditReconByNumOf` function and the `numOfRecons` config variable.

purgeByExpired

Uses the `deleteFromAuditReconByExpired` function and the config variables `intervalUnit` and `intervalValue`.

num-of-recons

The number of recon summary entries to keep for a given mapping, including all child entries.

intervalUnit

The type of time interval when using `purgeByExpired`. Acceptable values include: `minutes`, `hours`, or `days`.

intervalValue

The value of the time interval when using `purgeByExpired`. Set to an integer value.

Log file rotation

The file-based audit event handlers let you rotate audit log files, either automatically, based on a set of criteria, or by using a REST call.

To configure automatic log file rotation, set the following properties in your project's `audit.json` file:

```
{
    "class" : "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
    "config" : {
        "fileRotation" : {
            "rotationEnabled" : true,
            "maxFileSize" : 0,
            "rotationFilePrefix" : "",
            "rotationTimes" : [ ],
            "rotationFileSuffix" : "",
            "rotationInterval" : ""
        },
```

The file rotation properties are described in JSON Audit Event Handler Properties.

If you have enabled file rotation (`"rotationEnabled" : true`), you can rotate the JSON log files manually for a specific audit event topic, over REST. The following command saves the current access log file with a date and time stamp, then starts logging to a new file with the same base name.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/audit/access?handler=json&_action=rotate"
{
  "status": "OK"
}
```

If the command is successful, two `access.audit.json` files display in the `openidm/audit` directory, for example:

```
access.audit.json                       access.audit.json-2016.10.12-17.54.41
```

The file with the extension (`2016.10.12-17.54.41`) indicates that audit logging to this file ended on October 12, 2016, at 5:54:41 pm.

To configure log rotation in the admin UI, click **Configure > System Preferences > Audit**, and edit the JSON audit event handler (or the CSV audit event handler if you are logging to CSV). You can set all the log rotation properties on this screen.

Log file retention

Log file retention specifies how long audit files remain on disk before they are automatically deleted.

To configure log file retention, set the following properties in your project's `audit.json` file:

```
"fileRetention" : {
    "maxNumberOfHistoryFiles" : 100,
    "maxDiskSpaceToUse" : 1000,
    "minFreeSpaceRequired" : 10
},
```

The file retention properties are described in JSON Audit Event Handler Properties.

To configure log file retention in the admin UI, click **Configure > System Preferences > Audit**, and edit the JSON audit event handler (or the CSV audit event handler if you are logging to CSV). You can set all the log retention properties on this screen.

Query audit logs over REST

Regardless of where audit events are stored, they are accessible over REST on the `/audit` endpoint. The following sections describe how to query audit logs over REST.

> **Note**
>
> Queries on the audit endpoint must use queryFilter syntax.
> If you get no REST output on the correct endpoint, the corresponding audit file or JDBC table may not have audit data.
> Some examples in this section use client-assigned IDs (such as `bjensen` and `scarter`) when creating objects, because it makes the examples easier to read. If you create objects using the admin UI, they are created with server-assigned IDs (such as `55ef0a75-f261-47e9-a72b-f5c61c32d339`). Generally, immutable server-assigned UUIDs are used in production environments.

With the default audit configuration, reconciliation operations are not audited. To enable reconciliation logging, add `recon` to the list of audit topics for your event handler in `conf/audit.json`. For example:

```
"eventHandlers" : [
    {
        "class" : "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
        "config" : {
            "name" : "json",
            "logDirectory" : "&{idm.data.dir}/audit",
            "buffering" : {
                "maxSize" : 100000,
                "writeInterval" : "100 millis"
            },
            "topics" : [
                "access",
                "activity",
                "recon",
                "sync",
                "authentication",
                "config"
            ]
        }
    },
    {
        "class": "org.forgerock.openidm.audit.impl.RepositoryAuditEventHandler",
        "config": {
            "name": "repo",
            "enabled": true,
            "topics": [
                "access",
                "activity",
                "recon",
                "sync",
                "authentication",
                "config"
            ]
        }
    }
],
```

When enabled, the above example logs reconciliation operations in the file `/path/to/openidm/audit/recon.audit.json`, and in the repository. You can read and query the reconciliation audit logs over the REST interface, as outlined in the following examples.

To return all reconciliation operations logged in the audit log, query the `audit/recon` endpoint, as follows:

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

The following code extract shows the reconciliation audit log after the first reconciliation operation in the `sync-with-csv` sample. The output has been truncated for legibility.

```
{
   "result": [
     {
       "_id": "49bdb7cb-79a4-429d-856d-a7154005e41a-182",
       "transactionId": "49bdb7cb-79a4-429d-856d-a7154005e41a-177",
       "timestamp": "2017-02-28T13:07:20.487Z",
       "eventName": "recon",
       "userId": "openidm-admin",
       "exception": null,
       "linkQualifier": null,
       "mapping": "systemCsvfileAccounts_managedUser",
       "message": "Reconciliation initiated by openidm-admin",
       "sourceObjectId": null,
       "targetObjectId": null,
       "reconciling": null,
       "ambiguousTargetObjectIds": null,
       "reconAction": "recon",
       "entryType": "start",
       "reconId": "49bdb7cb-79a4-429d-856d-a7154005e41a-177"
     },
     {
       "_id": "49bdb7cb-79a4-429d-856d-a7154005e41a-192",
       "transactionId": "49bdb7cb-79a4-429d-856d-a7154005e41a-177",
       "timestamp": "2017-02-28T13:07:20.934Z",
       "eventName": "recon",
       "userId": "openidm-admin",
       "action": "CREATE",
       "exception": null,
       "linkQualifier": "default",
       "mapping": "systemCsvfileAccounts_managedUser",
       "message": null,
       "situation": "ABSENT",
       "sourceObjectId": "system/csvfile/account/scarter",
       "status": "SUCCESS",
       "targetObjectId": "managed/user/scarter",
       "reconciling": "source",
       "ambiguousTargetObjectIds": "",
       "entryType": "entry",
       "reconId": "49bdb7cb-79a4-429d-856d-a7154005e41a-177"
     },
     ...
 }
```


Most of the fields in the reconciliation audit log are self-explanatory. Each distinct reconciliation operation is identified by its `reconId`. Each entry in the log is identified by a unique `_id`. The first log entry indicates the status for the complete reconciliation operation. Successive entries indicate the status for each entry affected by the reconciliation.

To obtain information about a specific log entry, include its entry `_id` in the URL. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/audit/recon/414a4921-5d9d-4398-bf86-7d5312a9f5d1-146"
```

The following sample output shows the results of a read operation on a specific reconciliation audit entry. The entry shows the creation of scarter's account in the managed user repository, as the result of a reconciliation operation.


```
{
   "_id": "49bdb7cb-79a4-429d-856d-a7154005e41a-192",
   "transactionId": "49bdb7cb-79a4-429d-856d-a7154005e41a-177",
   "timestamp": "2017-02-28T13:07:20.934Z",
   "eventName": "recon",
   "userId": "openidm-admin",
   "action": "CREATE",
   "exception": null,
   "linkQualifier": "default",
   "mapping": "systemCsvfileAccounts_managedUser",
   "message": null,
   "situation": "ABSENT",
   "sourceObjectId": "system/csvfile/account/scarter",
   "status": "SUCCESS",
   "targetObjectId": "managed/user/scarter",
   "reconciling": "source",
   "ambiguousTargetObjectIds": "",
   "entryType": "entry",
   "reconId": "49bdb7cb-79a4-429d-856d-a7154005e41a-177"
 }
```


To obtain information for a specific reconciliation operation, include the `reconId` in the query. You can filter the log so that the query returns only the fields you want to see, by adding the `_fields` parameter.

The following query returns the `mapping`, `timestamp`, and `entryType` fields for a specific reconciliation operation:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/audit/recon?_queryFilter=/reconId+eq+"4261227f-1d44-4042-
ba7e-1dcbc6ac96b8"&_fields=mapping,timestamp,entryType'
{
  "result": [
    {
      "_id": "49bdb7cb-79a4-429d-856d-a7154005e41a-182",
      "mapping": "systemCsvfileAccounts_managedUser",
      "timestamp": "2017-02-28T13:07:20.487Z",
      "entryType": "start"
    },
    {
      "_id": "49bdb7cb-79a4-429d-856d-a7154005e41a-192",
      "mapping": "systemCsvfileAccounts_managedUser",
      "timestamp": "2017-02-28T13:07:20.934Z",
      "entryType": "entry"
    },
    {
      "_id": "49bdb7cb-79a4-429d-856d-a7154005e41a-191",
      "mapping": "systemCsvfileAccounts_managedUser",
      "timestamp": "2017-02-28T13:07:20.934Z",
      "entryType": "entry"
    },
    {
      "_id": "49bdb7cb-79a4-429d-856d-a7154005e41a-193",
      "mapping": "systemCsvfileAccounts_managedUser",
      "timestamp": "2017-02-28T13:07:20.943Z",
      "entryType": "summary"
    }
  ],
  ...
}
```

To query the reconciliation audit log for a particular reconciliation situation, include the `reconId` and the `situation` in the query. For example, the following query returns all ABSENT entries that were found during the specified reconciliation operation:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/audit/recon?_queryFilter=/reconId+eq+"414a4921-5d9d-4398-
bf86-7d5312a9f5d1-135"and+situation+eq"ABSENT"'
{
   "result": [
     {
       "_id": "49bdb7cb-79a4-429d-856d-a7154005e41a-192",
       "situation": "ABSENT",
       "reconId": "49bdb7cb-79a4-429d-856d-a7154005e41a-177",
       "transactionId": "49bdb7cb-79a4-429d-856d-a7154005e41a-177",
       "timestamp": "2017-02-28T13:07:20.934Z",
       "eventName": "recon",
       "userId": "openidm-admin",
       "action": "CREATE",
       "exception": null,
       "linkQualifier": "default",
       "mapping": "systemCsvfileAccounts_managedUser",
       "message": null,
       "sourceObjectId": "system/csvfile/account/scarter",
       "status": "SUCCESS",
       "targetObjectId": "managed/user/scarter",
       "reconciling": "source",
       "ambiguousTargetObjectIds": "",
       "entryType": "entry"
     },
     {
       "_id": "49bdb7cb-79a4-429d-856d-a7154005e41a-191",
       "situation": "ABSENT",
       "reconId": "49bdb7cb-79a4-429d-856d-a7154005e41a-177",
       "transactionId": "49bdb7cb-79a4-429d-856d-a7154005e41a-177",
       "timestamp": "2017-02-28T13:07:20.934Z",
       "eventName": "recon",
       "userId": "openidm-admin",
       "action": "CREATE",
       "exception": null,
       "linkQualifier": "default",
       "mapping": "systemCsvfileAccounts_managedUser",
       "message": null,
       "sourceObjectId": "system/csvfile/account/bjensen",
       "status": "SUCCESS",
       "targetObjectId": "managed/user/bjensen",
       "reconciling": "source",
       "ambiguousTargetObjectIds": "",
       "entryType": "entry"
     }
   ],
   ...
 }
```

The activity logs track all operations on internal (managed) and external (system) objects. Entries in the activity log contain identifiers for the reconciliation or synchronization action that triggered an activity, and for the original caller and the relationships between related actions.

You can access the activity logs over REST with the following call:

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

The following excerpt of the activity log shows the entries that created user `scarter`, with ID `42f8a60e-2019-4110-a10d-7231c3578e2b`:

```
{
   "result": [
     {
       "_id": "49bdb7cb-79a4-429d-856d-a7154005e41a-190",
       "transactionId": "49bdb7cb-79a4-429d-856d-a7154005e41a-177",
       "timestamp": "2017-02-28T13:07:20.894Z",
       "eventName": "activity",
       "userId": "openidm-admin",
       "runAs": "openidm-admin",
       "operation": "CREATE",
       "before": null,
       "after": {
         "mail": "scarter@example.com",
         "givenName": "Steven",
         "sn": "Carter",
         "description": "Created By CSV",
         "userName": "scarter",
         "password": {
           "$crypto": {
             "type": "x-simple-encryption",
             "value": {
               "cipher": "AES/CBC/PKCS5Padding",
               "salt": "tdrE2LZ+nBAnE44QY1UrCA==",
               "data": "P/z+OXA1x35aVWMRbOHMUQ==",
               "iv": "GACI5q4qZUWZRHzIle57TQ==",
               "key": "openidm-sym-default",
               "mac": "hqLmhjv67dxcmX8L3xxgZg=="
             }
           }
         },
         "telephoneNumber": "1234567",
         "accountStatus": "active",
         "effectiveRoles": [],
         "effectiveAssignments": [],
         "_rev": "00000000dc6160c8",
         "_id": "42f8a60e-2019-4110-a10d-7231c3578e2b"
       },
       "changedFields": [],
       "revision": "00000000bad8e88e",
       "message": "create",
       "objectId": "managed/user/42f8a60e-2019-4110-a10d-7231c3578e2b",
       "passwordChanged": true,
       "status": "SUCCESS"
     },
   ...
 }
```


For users who self-register through the End User UI, IDM provides more information. The following activity log excerpt depicts the information collected for user `jsanchez`. Note the following properties:

- IDM runs as user `anonymous`.
- Security questions (`kbaInfo`) are recorded with a salted hash SHA-256 algorithm.
- Marketing preferences are included.
- `termsAccepted` includes the date of the version of Terms & Conditions was accepted.

- The `message`, `context`, and `status` properties indicate that this user was created in the `SELFSERVICE` context, successfully.

```
{
     "_id" : "ddc7f35b-4b97-4586-be31-f5a2599b0764-10781",
     "transactionId" : "ddc7f35b-4b97-4586-be31-f5a2599b0764-10779",
     "timestamp" : "2017-07-26T17:14:24.137Z",
     "eventName" : "activity",
     "userId" : "anonymous",
     "runAs" : "anonymous",
     "operation" : "CREATE",
     "before" : null,
     "after" : {
       "kbaInfo" : [ {
         "answer" : {
           "$crypto" : {
             "value" : {
               "algorithm" : "SHA-256",
               "data" : "jENrBtzgIHscnOnvqSMYPTJKjZVVSN7XEfTp6VUpdXzNQsbCjmNQWpbfa1k1Zp24"
             },
             "type" : "salted-hash"
           }
         },
         "questionId" : "1"
       }, {
         "answer" : {
           "$crypto" : {
             "value" : {
               "algorithm" : "SHA-256",
               "data" : "obSQtsW3pgA4Yv4dPiISasvmrq4deoPOX4d9VRg+Bd/gGVDzu6fWPKd30Di3moEe"
             },
             "type" : "salted-hash"
           }
         },
         "questionId" : "2"
       } ],
       "userName" : "jsanchez",
       "givenName" : "Jane",
       "sn" : "Sanchez",
       "mail" : "jane.sanchez@example.com",
       "password" : {
         "$crypto" : {
           "type" : "x-simple-encryption",
           "value" : {
             "cipher" : "AES/CBC/PKCS5Padding",
             "stableId" : "openidm-sym-default",
             "salt" : "<hashValue>",
             "data" : "<encryptedValue>",
             "keySize" : 16,
             "purpose" : "idm.config.encryption",
             "iv" : "<encryptedValue>",
             "mac" : "<hashValue>"
           }
         }
       },
       "preferences" : {
         "updates" : true,
         "marketing" : false
       },
       "accountStatus" : "active",
       "effectiveRoles" : [ ],
       "effectiveAssignments" : [ ],
       "_rev" : "000000004eb36844",
```

```
        "_id" : "6e7fb8ce-4a97-42d4-90f1-b5808d51194a"
      },
      "changedFields" : [ ],
      "revision" : "000000004eb36844",
      "message" : "create",
      "context" : "SELFSERVICE",
      "objectId" : "managed/user/6e7fb8ce-4a97-42d4-90f1-b5808d51194a",
      "passwordChanged" : true,
      "status" : "SUCCESS"
    },
    ...
  }
```

To return the activity information for a specific action, include the `_id` of the action in the URL, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/audit/activity/414a4921-5d9d-4398-bf86-7d5312a9f5d1-145'
```

Each action in the activity log has a `transactionId` that is the same as the `transactionId` that was assigned to the incoming or initiating request. So, for example, if an HTTP request invokes a script that changes a user's password, the HTTP request is assigned a `transactionId`. The action taken by the script is assigned the same `transactionId`, which enables you to track the complete set of changes resulting from a single action. You can query the activity log for all actions that resulted from a specific transaction, by including the `transactionId` in the query.

The following command returns all actions in the activity log that occurred as a result of the reconciliation with the specified `transactionId`. The query results are restricted to only the `objectId` and the `resourceOperation`. You can see from the output that the reconciliation with this `transactionId` resulted in two CREATEs and two UPDATEs in the managed repository:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/audit/activity?_queryFilter=/transactionId+eq+"414a4921-5d9d-4398-
bf86-7d5312a9f5d1-135"&_fields=objectId,operation'
```

The following sample output shows the result of a query that created users scarter (with ID `42f8a60e-2019-4110-a10d-7231c3578e2b`) and bjensen (with ID `9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb`).

```
{
   "result" : [ {
     "_id" : "414a4921-5d9d-4398-bf86-7d5312a9f5d1-144",
     "objectId" : "managed/user/42f8a60e-2019-4110-a10d-7231c3578e2b",
     "operation" : "CREATE"
   }, {
     "_id" : "414a4921-5d9d-4398-bf86-7d5312a9f5d1-145",
     "objectId" : "managed/user/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
     "operation" : "CREATE"
   } ],
   "resultCount" : 2,
   "pagedResultsCookie" : null,
   "totalPagedResultsPolicy" : "NONE",
   "totalPagedResults" : -1,
   "remainingPagedResults" : -1
 }
```


For users who register through social identity providers, the following command returns JSON-formatted output for someone who has registered socially with a LinkedIn account, based on their `_id`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/audit/activity/b164fcb7-4a45-43b0-876d-083217254962'
```

The following output illustrates the data collected from a hypothetical LinkedIn user:

```
{
   "_id" : "94001c97-c597-46fa-a6c9-f53b0ddd7ff0-1982",
   "transactionId" : "94001c97-c597-46fa-a6c9-f53b0ddd7ff0-1974",
   "timestamp" : "2018-02-05T19:55:18.427Z",
   "eventName" : "activity",
   "userId" : "anonymous",
   "runAs" : "anonymous",
   "operation" : "CREATE",
   "before" : null,
   "after" : {
     "emailAddress" : "Xie@example.com",
     "firstName" : "Xie",
     "formattedName" : "Xie Na",
     "id" : "MW9FE_KyQH",
     "lastName" : "Na",
     "location" : {
       "country" : {
         "code" : "cn"
       },
       "name" : "Beijing, China"
     },
     "_meta" : {
       "subject" : "MW9FE_KyQH",
       "scope" : [ "r_basicprofile", "r_emailaddress" ],
       "dateCollected" : "2018-02-05T19:55:18.370"
     },
     "_rev" : "00000000c29c9f46",
     "_id" : "MW9FE_KyQH"
   },
   "changedFields" : [ ],
   "revision" : "00000000c29c9f46",
   "message" : "create",
   "provider" : "linkedIn",
   "context" : "SELFSERVICE",
   "objectId" : "managed/linkedIn/MW9FE_KyQH",
   "passwordChanged" : false,
   "status" : "SUCCESS"
 }
```


Note the `SELFSERVICE` context, which is included for all user self-registrations, either through the End User UI, or through a social identity provider.

LiveSync and implicit sync operations are logged in the file `/path/to/openidm/audit/sync.audit.json` and in the repository. You can read the synchronization audit logs over the REST interface, as outlined in the following examples.

To return all operations logged in the synchronization audit log, query the `audit/sync` endpoint, as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/audit/sync?_queryFilter=true"
{
   "result" : [ {
     "_id" : "53709f21-5b83-4ea0-ac35-9af39c3090cf-95",
     "transactionId" : "53709f21-5b83-4ea0-ac35-9af39c3090cf-85",
     "timestamp" : "2015-11-23T05:07:39.376Z",
     "eventName" : "sync",
     "userId" : "openidm-admin",
     "action" : "UPDATE",
     "exception" : null,
     "linkQualifier" : "default",
     "mapping" : "managedUser_systemLdapAccounts",
     "message" : null,
     "situation" : "CONFIRMED",
     "sourceObjectId" : "managed/user/128e0e85-5a07-4e72-bfc8-4d9500a027ce",
     "status" : "SUCCESS",
     "targetObjectId" : "uid=jdoe,ou=People,dc=example,dc=com"
   }, {
 ...
```

Most of the fields in the synchronization audit log are self-explanatory. Each entry in the log synchronization operation is identified by a unique `id`. Each _synchronization operation is identified with a `transactionId`. The same base `transactionId` is assigned to the incoming or initiating request — so if a modification to a user entry triggers an implicit synchronization operation, both the sync operation and the original change operation have the same `transactionId`. You can query the sync log for all actions that resulted from a specific transaction, by including the `transactionId` in the query.

To obtain information on a specific sync audit log entry, include its entry `_id` in the URL. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/audit/sync/53709f21-5b83-4ea0-ac35-9af39c3090cf-95"
{
  "_id" : "53709f21-5b83-4ea0-ac35-9af39c3090cf-95",
  "transactionId" : "53709f21-5b83-4ea0-ac35-9af39c3090cf-85",
  "timestamp" : "2015-11-23T05:07:39.376Z",
  "eventName" : "sync",
  "userId" : "openidm-admin",
  "action" : "UPDATE",
  "exception" : null,
  "linkQualifier" : "default",
  "mapping" : "managedUser_systemLdapAccounts",
  "message" : null,
  "situation" : "CONFIRMED",
  "sourceObjectId" : "managed/user/128e0e85-5a07-4e72-bfc8-4d9500a027ce",
  "status" : "SUCCESS",
  "targetObjectId" : "uid=jdoe,ou=People,dc=example,dc=com"
}
```

The authentication log includes details of all successful and failed authentication attempts. The output may be long. The output that follows is one excerpt from over 100 entries. To obtain the complete audit log over REST, use the following query:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/audit/authentication?_queryFilter=true"
...
    "principal" : [ "johndoe" ],
    "result" : "SUCCESSFUL",
    "userId" : "johndoe",
    "transactionId" : "cf967c5d-2b95-4cbe-9da0-e8952d726cd0-1016",
    "timestamp" : "2017-06-20T20:56:04.112Z",
    "eventName" : "LOGIN",
    "method" : "SOCIAL_PROVIDERS",
    "trackingIds" : [ "55fcec49-9631-4c00-83db-6931d10d04b8" ]
  }, {
    "_id" : "cf967c5d-2b95-4cbe-9da0-e8952d726cd0-1025",
    "provider" : "wordpress",
    "context" : {
      "component" : "managed/user",
      "provider" : "wordpress",
      "roles" : [ "internal/role/openidm-authorized" ],
      "ipAddress" : "172.16.201.36",
      "id" : "8ead23d1-4f14-4102-a130-c4093237f250",
      "moduleId" : "SOCIAL_PROVIDERS"
    },
    "entries" : [ {
      "moduleId" : "JwtSession",
      "result" : "SUCCESSFUL",
      "info" : {
        "org.forgerock.authentication.principal" : "johndoe"
      }
    } ],
...
```

The output depicts a successful login using Wordpress as a social identity provider. From the information shown, you can derive the following information:

- The `userId`, also known as the authentication `principal`, is `johndoe`. In the REST call that follows, you'll refer to how to use this information to filter authentication attempts made by that specific user.
- The login came from IP address `172.16.201.36`.
- The login used the `SOCIAL_PROVIDERS` authentication and the `JwtSession` session modules. For more information, refer to Authentication and Session Modules.

Login failures can also be instructive, as you'll refer to consecutive `moduleId` modules that correspond to the order of modules shown in your project's `authentication.json` file.

You can filter the results to return only those audit entries that you are interested in. For example, the following query returns all authentication attempts made by a specific user (`johndoe`), but displays only the security context and the result of the authentication attempt:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/audit/authentication?_queryFilter=/
principal+eq+"johndoe"&_fields=context,result'
{
  "result" : [ {
    "_id" : "cf967c5d-2b95-4cbe-9da0-e8952d726cd0-198",
    "provider" : null,
    "context" : {
      "ipAddress" : "172.16.201.36"
    },
    "entries" : [ {
      "moduleId" : "JwtSession",
      "result" : "FAILED",
      "reason" : { },
      "info" : { }
    },
...
  }, {
    "_id" : "cf967c5d-2b95-4cbe-9da0-e8952d726cd0-922",
    "provider" : "wordpress",
    "context" : {
      "component" : "null",
      "provider" : "wordpress",
      "roles" : [ "internal/role/openidm-authorized" ],
      "ipAddress" : "172.16.201.36",
      "id" : "e2b5bfc7-07a0-455c-a8f3-542089a8cc88",
      "moduleId" : "SOCIAL_PROVIDERS"
    },
     "entries" : [ {
      "moduleId" : "JwtSession",
      "result" : "FAILED",
      "reason" : { },
      "info" : { }
    },
 ...
    {
      "moduleId" : "SOCIAL_PROVIDERS",
      "result" : "SUCCESSFUL",
      "info" : {
        "org.forgerock.authentication.principal" : "johndoe"
      }
 ...
  }, {
    "_id" : "cf967c5d-2b95-4cbe-9da0-e8952d726cd0-1007",
    "provider" : "wordpress",
    "context" : {
      "component" : "managed/user",
      "provider" : "wordpress",
      "roles" : [ "internal/role/openidm-authorized" ],
      "ipAddress" : "172.16.201.36",
```

```
      "id" : "johndoe",
      "moduleId" : "SOCIAL_PROVIDERS"
    },
     "entries" : [ {
      "moduleId" : "JwtSession",
      "result" : "SUCCESSFUL",
      "info" : {
        "org.forgerock.authentication.principal" : "johndoe"
      }
...
```

The above excerpt illustrates a `FAILED` authentication attempt through a social identity provider, possibly based on a mistaken password. That is followed by a `SUCCESSFUL` authentication through the `SOCIAL_PROVIDERS` module, with the user included in the Managed User `component`.

This audit log lists changes made to the configuration in the audited server. You can read through the changes in the `config.extension` file in the `openidm/audit` directory.

You can also read the complete audit log over REST with the following query:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/audit/config?_queryFilter=true"
{
   "result" : [ {
     "_id" : "414a4921-5d9d-4398-bf86-7d5312a9f5d1-73",
     "operation" : "CREATE",
     "userId" : "openidm-admin",
     "runAs" : "openidm-admin",
     "transactionId" : "414a4921-5d9d-4398-bf86-7d5312a9f5d1-58",
     "revision" : null,
     "timestamp" : "2015-11-23T00:18:17.808Z",
     "objectId" : "ui",
     "eventName" : "CONFIG",
     "before" : "",
     "after" : "{ \"icons\":
     ...
      } ],
   "resultCount" : 3,
   "pagedResultsCookie" : null,
   "totalPagedResultsPolicy" : "NONE",
   "totalPagedResults" : -1,
   "remainingPagedResults" : -1
}
```

The output includes `before` and `after` entries, which represent the changes made to the configuration files.

View audit events in the admin UI

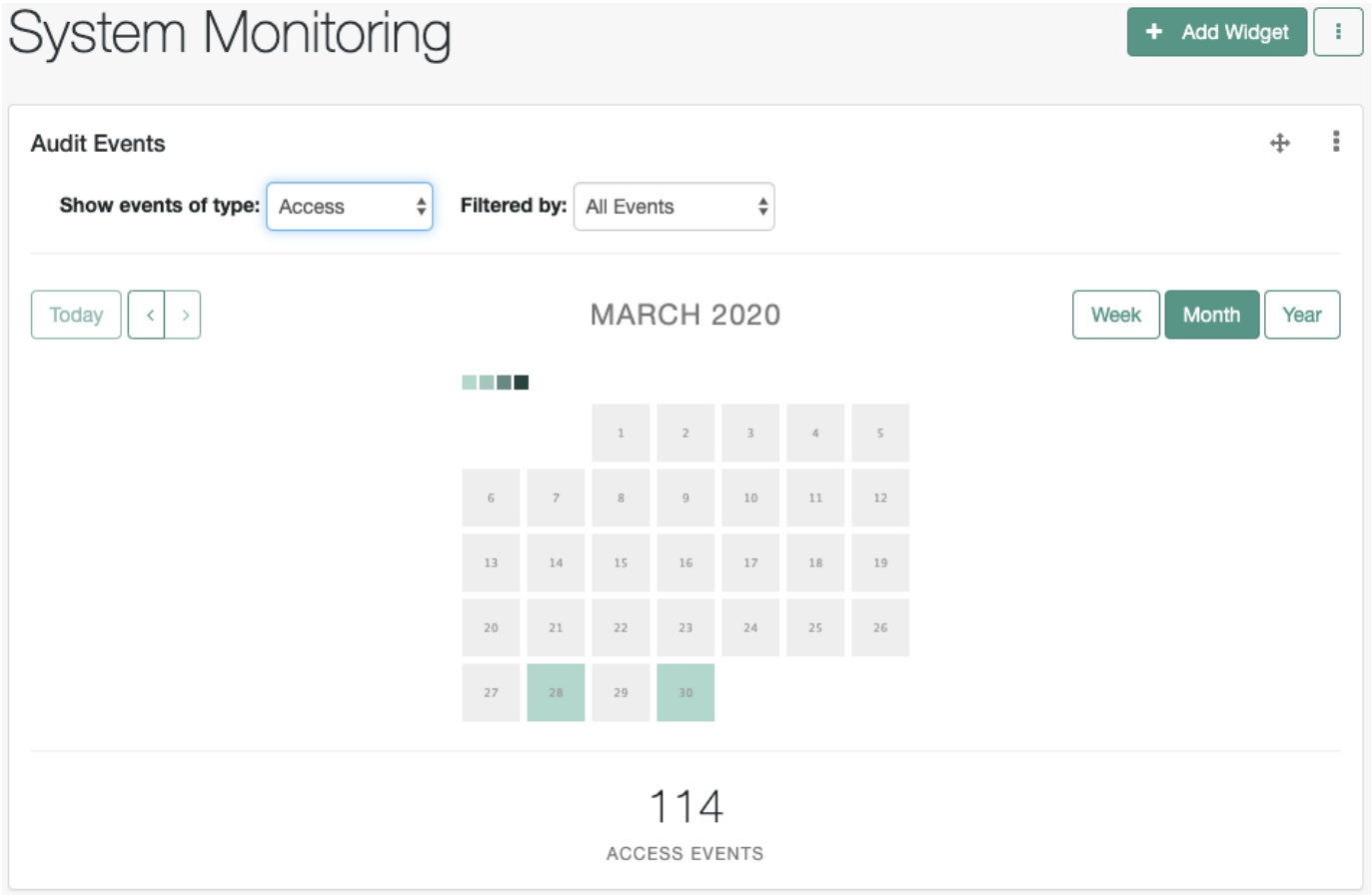
The admin UI includes an audit widget that provides a visual display of audit events.

The audit widget is displayed on the System Monitoring dashboard by default. To show audit events:

1. Log in to the admin UI, and select **Dashboards > System Monitoring**.
2. On the **Audit Events** widget, select the type of audit event that you want to view. The event types correspond to the audit event topics, described in Default Audit Event Topics.

   Depending on the event type, you filter the events further. For example, if you select **Config** as the event type, you can then select to view all configuration audit events, or only Creates, Reads, Updates, and so on.
3. By default, events are displayed for the current month. Use the arrow keys to scroll backwards and forwards to display the audit data for other months.

   The following image shows all access events for the month of March, 2020.



Use the move pointer to reposition the widget on the dashboard, or the vertical ellipses to delete the widget.

Audit log schema

The tables in this section show the schema for the six audit event topics. For the JSON audit event handler, each audit topic is logged to a distinct JSON file, with the topic in the filename. Files are created in the `openidm/audit` directory by default:

- `access.audit.json`
- `activity.audit.json`
- `authentication.audit.json`
- `config.audit.json`
- `recon.audit.json`
- `sync.audit.json`

You can parse the files in the `openidm/audit` directory using a JSON processor, such as `jq`. For example:

```
tail -f authentication.audit.json | jq .
{
  "context": {
    "component": "internal/user",
    "roles": [
      "internal/role/openidm-admin",
      "internal/role/openidm-authorized"
    ],
    "ipAddress": "0:0:0:0:0:0:0:1",
    "id": "openidm-admin",
    "moduleId": "INTERNAL_USER"
  },
  "entries": [
    {
      "moduleId": "JwtSession",
      "result": "SUCCESSFUL",
      "info": {
        "org.forgerock.authentication.principal": "openidm-admin"
      }
    }
  ],
  "principal": [
    "openidm-admin"
  ],
...
```

Reconciliation event topic properties

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

Event Property	Description
`transactionId`	UUID of the transaction; the same transaction might display for the same event in different audit event topics.
`timestamp`	The time that IDM logged the message, in UTC format; for example, `"2020-05-18T08:48:00.160Z"`.
`eventName`	Name of the audit event: `recon` for this log.
`userId`	User ID.
`trackingIds`	A unique value for an object being tracked.
`action`	Reconciliation action, shown as a Common REST action.
`exception`	Stack trace of the exception.
`linkQualifier`	Link qualifier applied to the action.
`mapping`	Name of the mapping used for the synchronization operation.
`message`	Description of the synchronization action.
`messageDetail`	Details from the synchronization run, shown as Common REST output.
`situation`	The synchronization situation.
`sourceObjectId`	The object ID on the source system, such as `managed/user/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb`.
`status`	Reconciliation result status, such as SUCCESS or FAILURE.
`targetObjectId`	The object ID on the target system, such as `system/csvfile/account/bjensen`.
`reconciling`	What is currently being reconciled, `source` for the first phase, `target` for the second phase.
`ambiguousTargetObjectIds`	When the `situation` is AMBIGUOUS or UNQUALIFIED, and IDM cannot distinguish between more than one target object, the object IDs are logged, to help figure out what was ambiguous.
`reconAction`	Reconciliation action, typically `recon` or `null`.
`entryType`	Type of reconciliation log entry, such as `start`, `entry`, or `summary`.
`reconId`	UUID for the reconciliation operation.

Synchronization event topic properties

Event Property	Description
`_id`	UUID for the message object, such as `"0419d364-1b3d-4e4f-b769-555c3ca098b0"`.
`transactionId`	UUID of the transaction; the same transaction might display for the same event in different audit event topics.
`timestamp`	Time that IDM logged the message, in UTC format; for example, `"2020-05-18T08:48:00.160Z"`.
`eventName`	Name of the audit event: `sync` for this log.
`userId`	User ID.
`trackingIds`	A unique value for an object being tracked.
`action`	The synchronization action, depicted as a Common REST action.
`exception`	Stack trace of the exception.
`linkQualifier`	Link qualifier applied to the action.
`mapping`	Name of the mapping used for the synchronization operation.
`message`	Description of the synchronization action.
`messageDetail`	Details from the reconciliation run, shown as REST output.
`situation`	The synchronization situation.
`sourceObjectId`	Object ID on the source system, such as `managed/user/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb`.
`status`	Reconciliation result status, such as SUCCESS or FAILURE.
`targetObjectId`	Object ID on the target system, such as `uid=jdoe,ou=People,dc=example,dc=com`.

Access event topic properties

Event Property	Description
`_id`	UUID for the message object, such as `"0419d364-1b3d-4e4f-b769-555c3ca098b0"`.
`timestamp`	Time that IDM logged the message, in UTC format; for example, `"2020-05-18T08:48:00.160Z"`.

Event Property	Description
`eventName`	Name of the audit event: `access` for this log.
`transactionId`	UUID of the transaction; the same transaction might display for the same event in different audit event topics.
`userId`	User ID.
`trackingId`	A unique value for the object being tracked.
`server.ip`	IP address of the IDM server.
`server.port`	Port number used by the IDM server.
`client.ip`	Client IP address.
`client.port`	Client port number.
`request.protocol`	Protocol for request, typically Common REST.
`request.operation`	Common REST operation taken on the object; for example, UPDATE, DELETE, or ACTION.
`request.detail`	Typically, details for an ACTION request.
`http.request.secure`	Boolean for request security.
`http.request.method`	HTTP method requested by the client.
`http.request.path`	Path of the HTTP request.
`http.request.queryParameters`	Parameters sent in the HTTP request, such as a key/value pair.
`http.request.headers`	HTTP headers for the request (optional).
`http.request.cookies`	HTTP cookies for the request (optional).
`http.response.headers`	HTTP response headers (optional).
`response.status`	Normally, SUCCESSFUL, FAILED, or null.
`response.statusCode`	SUCCESS in `response.status` leads to a null `response.statusCode`; FAILURE leads to a 400-level error.
`response.detail`	Message associated with `response.statusCode`, such as Not Found or Internal Server Error.
`response.elapsedTime`	Time to execute the access event.
`response.elapsedTimeUnits`	Units for response time.

Event Property	Description
`roles`	IDM roles associated with the request.

Activity event topic properties

Event Property	Description
`_id`	UUID for the message object, such as `"0419d364-1b3d-4e4f-b769-555c3ca098b0"`.
`timestamp`	Time that IDM logged the message, in UTC format; for example, `"2020-05-18T08:48:00.160Z"`.
`eventName`	Describes the audit event. Examples include `activity`, `workflow-complete_task`, and `relationship_created`.
`transactionId`	UUID of the transaction; the same transaction might display for the same event in different audit event topics.
`userId`	User ID.
`trackingId`	A unique value for the object being tracked.
`runAs`	User to run the activity as; may be used in delegated administration.
`objectId`	Object identifier, such as `/managed/user/42f8a60e-2019-4110-a10d-7231c3578e2b`.
`operation`	Common REST operation taken on the object; for example, UPDATE, DELETE, or ACTION.
`before`	JSON representation of the object prior to the activity.
`after`	JSON representation of the object after the activity.
`changedFields`	Fields that were changed, based on Fields to Watch.
`revision`	Object revision number.
`status`	Result, such as SUCCESS.
`message`	Human readable text about the action.
`passwordChanged`	True/False entry on changes to the password.
`context`	Flag for self-service logins, such as `SELFSERVICE`.
`provider`	Name of the self-service provider, usually a social identity provider.

Authentication event topic properties

Event Property	Description
`_id`	UUID for the message object, such as `"0419d364-1b3d-4e4f-b769-555c3ca098b0"`.
`timestamp`	Time that IDM logged the message, in UTC format; for example, `"2020-05-18T08:48:00.160Z"`.
`eventName`	Name of the audit event: `authentication` for this log.
`transactionId`	UUID of the transaction; the same transaction might display for the same event in different audit event topics.
`userId`	User ID.
`trackingId`	A unique value for the object being tracked.
`result`	Result of the transaction, either "SUCCESSFUL", or "FAILED".
`principal`	An array of the accounts used to authenticate, such as ["openidm-admin"].
`context`	The complete security context of the authentication operation, including the authenticating ID, targeted endpoint, authentication module, any roles applied, and the IP address from which the authentication request was made.
`entries`	JSON representation of the authentication session.
`method`	The authentication module used to authenticate, such as `JwtSession` or `MANAGED_USER`.
`provider`	Social identity provider name.

Configuration event topic properties

Event Property	Description
`_id`	UUID for the message object, such as `"0419d364-1b3d-4e4f-b769-555c3ca098b0"`.
`timestamp`	Time that IDM logged the message, in UTC format; for example, `"2020-05-18T08:48:00.160Z"`.
`eventName`	Name of the audit event: `config` for this log.
`transactionId`	UUID of the transaction; the same transaction might display for the same event in different audit event topics.
`userId`	User ID.

Event Property	Description
`trackingId`	A unique value for the object being tracked.
`runAs`	User to run the activity as; can be used in delegated administration.
`objectId`	Object identifier, such as `ui`.
`operation`	Common REST operation taken on the object; for example, UPDATE, DELETE, or ACTION.
`before`	JSON representation of the object prior to the activity.
`after`	JSON representation of the object after to the activity.
`changedFields`	Fields that were changed, based on Fields to Watch.
`revision`	Object revision number.

Audit event handler configuration

To configure an audit event handler, set the `config` properties for that handler in your project's `conf/audit.json` file.

To configure these properties from the admin UI, click **Configure > System Preferences > Audit**, and click the edit icon for your event handler.

The tables in this section show the configuration properties common to all audit event handlers, then the properties specific to each audit event handler.

Common audit event handler properties

UI Label / Text	audit.json File Label	Description
Name	`name`	`config` sub-property. The name of the audit event handler.
Audit Events	`topics`	`config` sub-property; the list of audit topics that are logged by this audit event handler, for example, `access`, `activity`, and `config`.
Use for Queries	`handlerForQueries`	Specifies whether this audit event handler manages the queries on audit logs.
Enabled	`enabled`	`config` sub-property; specifies whether the audit event handler is enabled. An audit event handler can be configured, but disabled; in which case, it will not log events.
n/a	`config`	The JSON object used to configure the handler; includes several sub-properties.

UI Label / Text	audit.json File Label	Description
Shown only in `audit.json`	`class`	The class name in the Java file(s) used to build the handler.

JSON audit event handler properties

Property	Description
`fileRotation`	Groups the file rotation configuration parameters.
`rotationEnabled`	Specifies whether file rotation is enabled. Boolean: true, or false.
`maxFileSize`	The maximum size of an audit file, in bytes, before rotation is triggered.
`rotationFilePrefix`	The prefix to add to the start of an audit file name when it is rotated.
`rotationTimes`	Specifies a list of times when file rotation should be triggered. The times must be provided as durations, offset from midnight. For example, a list of `10 minutes, 20 minutes, 30 minutes` will cause files to rotate at 10, 20 and 30 minutes after midnight.
`rotationFileSuffix`	The suffix appended to rotated audit file names. This suffix should take the form of a timestamp, in simple date format. The default suffix format, if none is specified, is `-yyyy.MM.dd-HH.mm.ss`.
`rotationInterval`	The interval to trigger a file rotation, expressed as a duration. For example, `5 seconds`, `5 minutes`, `5 hours`. A value of `0` or `disabled` disables time-based file rotation. Note that you can specify a list of `rotationTimes` and a `rotationInterval`. The audit event handler checks all rotation and retention policies on a periodic basis, and assesses whether each policy should be triggered at the current time, for a particular audit file. The first policy to meet the criteria is triggered.
`fileRetention`	Groups the file retention configuration parameters. The retention policy specifies how long audit files remain on disk before they are automatically deleted.
`maxNumberOfHistoryFiles`	The maximum number of historical audit files that can be stored. If the total number of audit files exceeds this maximum, older files are deleted. A value of `-1` disables purging of old log files.
`maxDiskSpaceToUse`	The maximum disk space, in bytes, that can be used for audit files. If the total space occupied by the audit files exceeds this maximum, older files are deleted. A negative or zero value indicates that this policy is disabled; that is, that unlimited disk space can be used for historical audit files.

Property	Description
`minFreeSpaceRequired`	The minimum free disk space, in bytes, required on the system that houses the audit files. If the free space drops below this minimum, older files are deleted. A negative or zero value indicates that this policy is disabled; that is, that no minimum space requirements apply.
`rotationRetentionCheckInterval`	Interval for periodically checking file rotation and retention policies. The interval must be a duration; for example, `5 seconds`, `5 minutes`, or `5 hours`.
`logDirectory`	Directory with JSON audit files
`elasticsearchCompatible`	Enable ElasticSearch JSON format compatibility. Boolean, true or false. Set this property to `true`, for example, if you are using Logstash to feed into ElasticSearch. When `elasticsearchCompatible` is `true`, the handler renames the `_id` field to `_eventId` because `_id` is reserved by ElasticSearch. The rename is reversed after JSON serialization, so that other handlers can safely use the original field name. For more information, refer to the ElasticSearch documentation.
`buffering`	Configuration for event buffering.
`maxSize`	The maximum number of events that can be buffered (default/minimum: 100000).
`writeInterval`	The delay after which the file-writer thread is scheduled to run after encountering an empty event buffer (units of 'ms' are recommended). Default: 100 ms.

CSV audit event handler properties

UI Label / Text	audit.json File Label	Description
File Rotation	`fileRotation`	Groups the file rotation configuration parameters.
rotationEnabled	`rotationEnabled`	Specifies whether file rotation is enabled. Boolean: true, or false.
maxFileSize	`maxFileSize`	The maximum size of an audit file, in bytes, before rotation is triggered.
rotationFilePrefix	`rotationFilePrefix`	The prefix to add to the start of an audit file name when it is rotated.

UI Label / Text	audit.json File Label	Description
Rotation Times	`rotationTimes`	Specifies a list of times when file rotation should be triggered. The times must be provided as durations, offset from midnight. For example, a list of `10 minutes, 20 minutes, 30 minutes` will cause files to rotate at 10, 20 and 30 minutes after midnight.
File Rotation Suffix	`rotationFileSuffix`	The suffix appended to rotated audit file names. This suffix should take the form of a timestamp, in simple date format. The default suffix format, if none is specified, is `-yyyy.MM.dd-HH.mm.ss`.
Rotation Interval	`rotationInterval`	The interval to trigger a file rotation, expressed as a duration. For example, `5 seconds`, `5 minutes`, `5 hours`. A value of `0` or `disabled` disables time-based file rotation. Note that you can specify a list of `rotationTimes` and a `rotationInterval`. The audit event handler checks all rotation and retention policies on a periodic basis, and assesses whether each policy should be triggered at the current time, for a particular audit file. The first policy to meet the criteria is triggered.
File Retention	`fileRetention`	Groups the file retention configuration parameters. The retention policy specifies how long audit files remain on disk before they are automatically deleted.
Maximum Number of Historical Files	`maxNumberOfHistoryFiles`	The maximum number of historical audit files that can be stored. If the total number of audit files exceeds this maximum, older files are deleted. A value of `-1` disables purging of old log files.
Maximum Disk Space	`maxDiskSpaceToUse`	The maximum disk space, in bytes, that can be used for audit files. If the total space occupied by the audit files exceeds this maximum, older files are deleted. A negative or zero value indicates that this policy is disabled; that is, that unlimited disk space can be used for historical audit files.
Minimum Free Space Required	`minFreeSpaceRequired`	The minimum free disk space, in bytes, required on the system that houses the audit files. If the free space drops below this minimum, older files are deleted. A negative or zero value indicates that this policy is disabled; that is, that no minimum space requirements apply.
rotationRetentionCheckInterval	`rotationRetentionCheckInterval`	Interval for periodically checking file rotation and retention policies. The interval must be a duration; for example, `5 seconds`, `5 minutes`, or `5 hours`.

UI Label / Text	audit.json File Label	Description
Log Directory	`logDirectory`	Directory with CSV audit files.
CSV Output Formatting	`formatting`	
quoteChar	`quoteChar`	Formatting: Character used around a CSV field.
delimiterChar	`delimiterChar`	Formatting: Character between CSV fields.
End of Line Symbols	`endOfLineSymbols`	Formatting: end of line symbol, such as `\n` or `\r`.
Security: CSV Tamper Evident Configuration	`security`	Uses keystore-based signatures.
Enabled	`enabled`	CSV Tamper Evident Configuration: true, or false.
Filename	`filename`	CSV Tamper Evident Configuration: Path to the Java keystore.
Password	`password`	CSV Tamper Evident Configuration: Password for the Java keystore.
Keystore Handler	`keystoreHandlerName`	CSV Tamper Evident Configuration: Keystore name. The value of this property must be `openidm`. This is the name that the audit service provides to the ForgeRock Common Audit Framework for the configured IDM keystore.
Signature Interval	`signatureInterval`	CSV Tamper Evident Configuration: Signature generation interval. Default = 1 hour. Units described in Restrictions on Configuring the CSV Audit Handler in the UI.
Buffering	`buffering`	Configuration for optional event buffering.
enabled	`enabled`	Buffering: true, or false.
autoFlush	`autoFlush`	Buffering: avoids flushing after each event.

Repository and router audit event handler properties

In addition to the common properties, the Repository and Router audit event handlers both have one unique property, `resourcePath`:

```
{
    "class" : "org.forgerock.openidm.audit.impl.RouterAuditEventHandler",
    "config" : {
        "name" : "router",
        "topics" : [ "access", "activity", "sync", "authentication", "config" ],
        "resourcePath" : "system/auditdb"
    }
},
```

UI Label / Text	audit.json File Label	Description
resourcePath	`resourcePath`	Path to the repository resource.

JMS audit event handler properties

Note that the JMS audit handler `config` in `audit.json` includes the ForgeRock audit event topics *and* JMS audit topics.

To use the JMS resources provided by your web application container, leave the `JNDI Context Properties` settings empty. Values for `topicName` and `connectionFactoryName` will then depend on the configuration of your web application container.

UI Label / Text	audit.json File Label	Description
Delivery Mode	`deliveryMode`	Required property, for messages from a JMS provider; may be `PERSISTENT` or `NON_PERSISTENT`
Session Mode	`sessionMode`	Acknowledgement mode, in sessions without transactions. May be `AUTO`, `CLIENT`, or `DUPS_OK`.
Batch Configuration Settings	`batch`	Options for batch messaging.
Write Interval	`writeInterval`	Interval at which buffered events are written to JMS (units of 'ms' or 's' are recommended). Default is 10 ms.
Capacity	`capacity`	Maximum event count in the batch queue; additional events are dropped.
Maximum Batched Events	`maxBatchedEvents`	Maximum number of events per batch.
JNDI Configuration	`jndiConfiguration`	Java Naming and Directory Interface (JNDI) Configuration Settings.
JNDI Context Properties	`contextProperties`	Settings to populate the JNDI initial context with.
JNDI Context Factory	`java.naming.factory.initial`	Initial JNDI context factory, such as `com.tibco.tibjms.naming.TibjmsInitialContextFactory`.

UI Label / Text	audit.json File Label	Description
JNDI Provider URL	`java.naming.provider.url`	Depends on provider; options include `tcp://localhost:61616` and `tibjmsnaming://192.168.1.133:7222`.
JNDI Topic	`topic.forgerock.idm.audit`	Relevant JNDI topic; default=`forgerock.idm.audit`.
JNDI Topic Name	`topicName`	JNDI lookup name for the JMS topic.
Connection Factory	`connectionFactoryName`	JNDI lookup name for the JMS connection factory.

Syslog audit event handler properties

UI Label / Text	audit.json File Label	Description
protocol	`protocol`	Transport protocol for Syslog messages; may be `TCP` or `UDP`.
host	`host`	Host name or IP address of the receiving Syslog server.
port	`port`	The TCP/IP port number of the receiving Syslog server.
connectTimeout	`connectTimeout`	Timeout for connecting to the Syslog server (seconds).
facility	`facility`	Options shown in the admin UI, `KERN`, `USER`, `MAIL`, `DAEMON`, `AUTH`, `SYSLOG`, `LPR`, `NEWS`, `UUCP`, `CRON`, `AUTPRIV`, `FTP`, `NTP`, `LOGAUDIT`, `LOGALERT`, `CLOCKD`, `LOCAL0`, `LOCAL1`, `LOCAL2`, `LOCAL3`, `LOCAL4`, `LOCAL5`, `LOCAL6`, `LOCAL7` correspond directly to facility values shown in RFC 5424 - The Syslog Protocol.
SeverityFieldMappings	`severityFieldMappings`	Sets the correspondence between audit event fields and Syslog severity values.
topic	`topic`	Severity Field Mappings: the audit event topic to which the mapping applies.
field	`field`	Severity Field Mappings: the audit event field to which the mapping applies; taken from the JSON schema for the audit event content.
Value Mappings	`valueMappings`	Severity Field Mappings: The map of audit event values to Syslog severities. Syslog severities may be: EMERGENCY, ALERT, CRITICAL, ERROR, WARNING, NOTICE, INFORMATIONAL, or DEBUG, in descending order of importance.

UI Label / Text	audit.json File Label	Description
Buffering	`buffering`	Disabled by default; all messages written immediately to the log.

Configure notifications

The customizable notification service sends messages, based on changes to objects. The notification service uses filters to assess incoming requests. If the filter conditions are met, the service sends the corresponding notification. Notification messages are sent to whatever routes you specify.

In a JDBC repository, notifications are stored in the `notificationobjects` table. The `notificationobjectproperties`, serves as the index table. In a DS repository, notifications are stored under the DN `"ou=notification,ou=internal,dc=openidm,dc=forgerock,dc=com"`.

The notification service is disabled by default. To enable the service, add `openidm.notifications=true` to your project's `resolver/boot.properties` file. You can perform additional configuration using the `conf/notificationFactory.json` file.

```
{
    "enabled" :{
        "$bool" : "&{openidm.notifications|false}"
    },
    "threadPool" : {
        "steadyPoolThreads" : 1,
        "maxPoolThreads" : 2,
        "threadKeepAlive" : 60,
        "maxQueueSize" : 20000
    }
}
```

Important

Changing the notifications thread pool settings can adversely affect performance.

Notifications for a managed object are injected into a property in that object. The name of this property is specified in the managed object schema, in `conf/managed.json`. For example, notifications for managed user objects rely on the following construct in the `user` object definition in `managed.json`:

```
{
    "objects" : [
        {
            "name" : "user",
            ...
            "notifications" : {
                "property" : "_notifications"
            },
            ...
        },
        ...
    ]
}
```

This excerpt indicates that notifications are injected into the `_notifications` property of the user object by default. The `notifications` object is mandatory for notifications to be generated for that managed object type. However, you can change the name of the property that is injected into the managed object when notifications are generated. If you omit the `property` field from the `notifications` object, notifications are stored in the `_notifications` field by default.

Important

- The ability to tie a specific notification to its corresponding managed object is regarded as an *internal object relation*. Notifications are therefore also configured in `conf/internal.json` with the following object:

  ```
  {
      "name" : "notification",
      "properties" : {
          "target" : {
              "reversePropertyName" : "_notifications"
          }
      }
  }
  ```

  If you change the `property` field in `managed.json` to something other than `_notifications`, you must also update the corresponding `reversePropertyName` in `internal.json` to reflect the change.

  Note

  The internal object service does not support runtime changes. If you update `conf/internal.json` over REST, you must restart IDM for the change to take effect.

- If you have configured notifications for more than one managed object type, all the object types must use the same notification property name.

Custom notifications

Notifications are configured in files named `notification-event.json`, where event refers to the event that triggers the notification.

By default, IDM doesn't send any notifications for password or profile updates. To enable these notifications, add the applicable line to your project's `resolver/boot.properties` file:

- `openidm.notifications.passwordUpdate=true`
- `openidm.notifications.profileUpdate=true`

These notifications are configured in the `conf/notification-passwordUpdate.json` and `conf/notification-profileUpdate.json` files, respectively. You can use these default notification configuration files as the basis for setting up custom notifications.

The default `notification-passwordUpdate.json` file shows the structure of a notification configuration:

```
{
    "enabled" : {
        "$bool" : "&{openidm.notifications.passwordUpdate|false}"
    },
    "path" : "managed/user/*",
    "methods" : [
        "update",
        "patch"
    ],
    "condition" : {
        "type" : "groovy",
        "globals" : {
            "propertiesToCheck" : [
                "password"
            ]
        },
        "file" : "propertiesModifiedFilter.groovy"
    },
    "target" : {
        "resource" : "managed/user/{{response/_id}}"
    },
    "notification" : {
        "notificationType": "info",
        "message": "Your password has been updated."
    }
}
```

`enabled` *boolean, true or false*

Specifies whether notifications will be triggered for that configured event. To enable/disable, set the `openidm.notifications.passwordUpdate` property in the `resolver/boot.properties` file.

`path` *string*

Specifies where the filter listens on the router. For user notifications, this is typically `managed/user/*`.

`methods` *array of strings (optional)*

One or more ForgeRock REST verbs, specifying the actions that should trigger the notification. These can include `create`, `read`, `update`, `delete`, `patch`, `action`, and `query`. If no `methods` are specified, the default is to listen for all methods.

`condition` *string or object*

An inline script or a path to a script `file` that specifies the condition on which the notification is triggered. The `passwordUpdate` notification configuration references the groovy script, `/path/to/openidm/bin/defaults/script/propertiesModifiedFilter.groovy`. This script monitors the properties listed in the `propertiesToCheck` array, and sends a notification when those properties are changed. The script also checks whether a modified property is the child (or parent) of a watched property.

To specify additional properties to watch, add the property names to the array of `propertiesToCheck`. The properties that you can specify here are limited to existing user properties defined in your `managed.json` file. For example, the following excerpt of the `notification-profileUpdate.json` file shows the properties that will trigger notifications if their values are changed:

```
...
    "condition" : {
        "type" : "groovy",
        "globals" : {
            "propertiesToCheck" : [
                "userName",
                "givenName",
                "sn",
                "mail",
                "description",
                "accountStatus",
                "telephoneNumber",
                "postalAddress",
                "city",
                "postalCode",
                "country",
                "stateProvince",
                "preferences"
            ]
        },
        "file" : "propertiesModifiedFilter.groovy"
    },
...
```

`target` *object*

The target resource to which notifications are sent, typically `managed/user/{{response/_id}}`.

The `target.resource` field supports `{{token}}` replacement with contextual variables. The following variables are in scope:

- `request`
- `context`
- `resourceName`
- `response`

`notification`

The actual notification, including the `notificationType` (`info`, `warning`, or `error`) and the `message` that is sent to the user.

The `notification.message` field supports `{{token}}` replacement with contextual variables, as described previously for `target.resource`.

Notification configuration files follow the format of the `router.json` file. For more information about how filtering is configured in `router.json`, refer to Router configuration.

Additional sample notification configuration files can be found in the `/path/to/openidm/samples/example-configurations/conf` directory:

`notification-newReport.json`

This configuration notifies managers when a new direct reporting employee is assigned to them.

`notification-termsUpdate.json`

This configuration notifies all users who have accepted the Terms and Conditions of any updates to those Terms and Conditions.

To use these files (or create your own notifications based on these files), copy them to your project's `conf/` directory.

Limits on notification endpoints

Although notifications are highly configurable, you cannot apply them to services with their own internal routers, including internal objects. This list includes:

`workflow/taskinstance`
`workflow/processdefinition`
`workflow/processinstance`
`metrics/api`
`metrics/prometheus`
`scheduler/job`
`scheduler/trigger`
`scheduler/waitingTriggers`
`scheduler/acquiredTriggers`
`info/ping`
`info/login`
`info/version`
`info/uiconfig`
`info/features`
`internal/{object}`
`internal/{object}/{object_id}/relationship`
`managed/{object}/{object_id}/relationship`

Schedules

Guide to configuring schedules and scanning tasks.

This guide covers schedules and scanning tasks.

Scheduling

Schedule tasks and events.

Task Scanner

Scan data to trigger tasks.

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

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

Schedule tasks and events

The scheduler service lets you schedule reconciliation and synchronization tasks, trigger scripts, collect and run reports, trigger workflows, and perform custom logging.

The service depends on the Quartz Scheduler (bundled with IDM), and supports Quartz simple triggers and cron triggers. Use the trigger type that suits your scheduling requirements. For more information, refer to the Quartz documentation on SimpleTriggers and CronTriggers.

By default, IDM picks up changes to scheduled tasks and events dynamically, during initialization and also at runtime. For more information, refer to Configuration changes.

In addition to the fine-grained scheduling facility, you can perform a scheduled batch scan for a specified date in IDM data, and then automatically run a task when this date is reached. For more information, refer to Scan data to trigger tasks.

Configure the scheduler service

There is a distinction between the configuration of the scheduler service, and the configuration of individual scheduled tasks and events. The scheduler configuration has the following format, and configures the Quartz Scheduler:

```
{
    "threadPool" : {
        "threadCount" : 10
    },
    "scheduler" : {
        "executePersistentSchedules" : {
            "$bool" : "&{openidm.scheduler.execute.persistent.schedules}"
        }
    }
}
```

- `threadCount` specifies the maximum number of threads that are available for running scheduled tasks concurrently.
- `executePersistentSchedules` lets you disable persistent schedules for a specific node. If this parameter is set to `false`, the Scheduler Service will support the management of persistent schedules (CRUD operations) but it will not run any persistent schedules. The value of this property can be a string or boolean. Its default value (set in `resolver/boot.properties`) is `true`.
- `advancedProperties` (optional) lets you configure additional properties for the Quartz Scheduler.

For details of all the configurable properties for the Quartz Scheduler, refer to the Quartz Scheduler Configuration Reference.

Configure schedules

You can schedule tasks and events using:

- Admin UI
- REST

By convention, IDM uses file names of the form `schedule-schedule-name.json`, where schedule-name is a logical name for the scheduled operation; for example, `schedule-reconcile_systemCsvAccounts_managedUser.json`. There are several example schedule configuration files in the `openidm/samples/example-configurations/schedules` directory.

Each schedule configuration has the following format:

```
{
 "enabled"             : boolean,
 "persisted"           : boolean,
 "recoverable"         : boolean,
 "concurrentExecution" : boolean,
 "type"                : "simple | cron",
 "repeatInterval"      : (optional) integer,
 "repeatCount"         : (optional) integer,
 "startTime"           : "(optional) time",
 "endTime"             : "(optional) time",
 "schedule"            : "cron expression",
 "misfirePolicy"       : "optional, string",
 "invokeService"       : "service identifier",
 "invokeContext"       : "service specific context info",
 "invokeLogLevel"      : "(optional) level"
}
```

Schedule configuration properties

The schedule configuration properties are defined as follows:

`enabled`

Set to `true` to enable the schedule. When this property is `false`, IDM considers the schedule configuration dormant, and does not allow it to be triggered or launched.

If you want to retain a schedule configuration, but do not want it used, set `enabled` to `false` for task and event schedulers, instead of changing the configuration or `cron` expressions.

`persisted` *(optional)*

Specifies whether the schedule state should be persisted or stored *only* in RAM. Boolean (`true` or `false`), `true` by default.

In a clustered environment, this property must be set to `true` to have the schedule fire only once across the cluster. For more information, refer to Configure Persistent Schedules.

> **Note**
>
> If the schedule is stored only in RAM, the schedule will be lost when IDM is restarted.

`recoverable` *(optional)*

Specifies whether jobs that have failed mid-execution (as a result of a JVM crash or otherwise unexpected termination) should be recovered. Boolean (`true` or `false`), `false` by default.

`concurrentExecution`

Specifies whether multiple instances of the same schedule can run concurrently. Boolean (`true` or `false`), `false` by default. Multiple instances of the same schedule cannot run concurrently by default. This setting prevents a new scheduled task from being launched before the same previously launched task has completed. For example, under normal circumstances you would want a liveSync operation to complete before the same operation was launched again. To enable multiple schedules to run concurrently, set this parameter to `true`. The behavior of missed scheduled tasks is governed by the misfire policy.

`type`

The trigger type, either `simple` or `cron`.

To decide which trigger type to use, refer to the Quartz documentation on SimpleTriggers and CronTriggers.

`repeatCount`

Used only for simple triggers (`"type" : "simple"`).

The number of times the schedule must be repeated. The repeat count can be zero, a positive integer, or -1. A value of -1 indicates that the schedule should repeat indefinitely.

If you do not specify a repeat count, the value defaults to -1.

`repeatInterval`

Used only for simple triggers (`"type" : "simple"`).

Specifies the interval, in milliseconds, between trigger firings. The repeat interval must be zero or a positive long value. If you set the repeat interval to zero, the scheduler will trigger `repeatCount` firings concurrently (or as close to concurrently as possible).

If you do not specify a repeat interval, the value defaults to 0.

`startTime` *(optional)*

This parameter starts the schedule at some time in the future. If the parameter is omitted, empty, or set to a time in the past, the task or event is scheduled to start immediately.

Use ISO 8601 format to specify times and dates (`yyyy-MM-dd'T'HH:mm:ss`).

To specify a time zone, include the time zone at the end of the `startTime` , in the format `+|-hh:mm` ; for example `2017-10-31T15:53:00+05:00` . If you specify both a `startTime` and an `endTime` , they must have the same time zone.

`endTime` *(optional)*

Specifies when the schedule must end, in ISO 8601 format (`yyyy-MM-dd'T'HH:mm:ss+|-hh:mm`).

`schedule`

Used only for cron triggers (`"type" : "cron"`).

Takes `cron` expression syntax. For more information, refer to the CronTrigger Tutorial and Lesson 6: CronTrigger.

`misfirePolicy`

For persistent schedules, this optional parameter specifies the behavior if the scheduled task is missed, for some reason. Possible values are as follows:

- `fireAndProceed` . The first run of a missed schedule is immediately launched when the server is back online. Subsequent runs are discarded. After this, the normal schedule is resumed.
- `doNothing` . All missed schedules are discarded and the normal schedule is resumed when the server is back online.

`invokeService`

Defines the type of scheduled event or action. The value of this parameter can be one of the following:

- `sync` for reconciliation.
- `provisioner` for liveSync.
- `script` to call some other scheduled operation defined in a script.
- `taskScanner` to define a scheduled task that queries a set of objects. For more information, refer to Scan data to trigger tasks.

`invokeContext`

Specifies contextual information, depending on the type of scheduled event (the value of the `invokeService` parameter).

The following example invokes reconciliation:

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

The following example invokes a liveSync operation:

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

For scheduled liveSync tasks, the `source` property follows IDM's convention for a pointer to an external resource object and takes the form `system/resource-name/object-type`.

The following example invokes a script, which prints the node ID performing the scheduled job and the time to the console.

A similar sample schedule is provided in `schedule-script.json` in the `/path/to/openidm/samples/example-configurations/schedules` directory.

```
{
    "enabled" : true,
    "type": "simple",
    "repeatInterval": 3600000,
    "persisted" : true,
    "concurrentExecution" : false,
    "invokeService": "script",
    "invokeContext": {
        "script" : {
            "type" : "text/javascript",
            "source" : "java.lang.System.out.println('Job executing on ' +
identityServer.getProperty('openidm.node.id') + ' at: ' + java.lang.System.currentTimeMillis());"
        }
    }
}
```

> **Note**
>
> These are sample configurations only. Your schedule configuration will differ according to your specific requirements.

`invokeLogLevel` *(optional)*

Specifies the level at which the invocation will be logged. Particularly for schedules that run very frequently, such as liveSync, the scheduled task can generate significant output to the log file, and you should adjust the log level accordingly. The default schedule log level is `info`. The value can be set to any one of the SLF4J log levels:

- `trace`
- `debug`
- `info`
- `warn`
- `error`
- `fatal`

Manage schedules using REST

The scheduler service is exposed under the `/openidm/scheduler` context path. Within this context path, the defined scheduled jobs are accessible at `/openidm/scheduler/job`. A job is the actual task that is run. Each job contains a *trigger* that starts the job. The trigger defines the schedule according to which the job is executed. You can read and query the existing triggers on the `/openidm/scheduler/trigger` context path.

The following examples show how schedules are validated, created, read, queried, updated, and deleted, over REST, by using the scheduler service.

> **Note**
>
> When you configure schedules over REST, changes made to the schedules are not pushed back into the configuration service. Managing schedules by using the `/openidm/scheduler/job` context path essentially bypasses the configuration service and sends the request directly to the scheduler.
> If you need to perform an operation on a schedule that was created by using the configuration service (by placing a schedule file in the `conf/` directory), you must direct your request to the `/openidm/config` context path, and not to the `/openidm/scheduler/job` context path.
> PATCH operations are not supported on the `scheduler` context path. To patch a schedule, use the `config` context path.

Validate cron trigger expressions

Schedules are defined using Quartz cron or simple triggers. If you use a cron trigger, you can validate your cron expression by sending the expression as a JSON object to the `scheduler` context path:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0" \
--request POST \
--data '{
    "cronExpression": "0 0/1 * * * ?"
}' \
"http://localhost:8080/openidm/scheduler/job?_action=validateQuartzCronExpression"
{
  "valid": true
}
```

Define a schedule

To define a new schedule, send a PUT or POST request to the `scheduler/job` context path with the details of the schedule in the JSON payload. A PUT request lets you specify the ID of the schedule. A POST request assigns an ID automatically.

The following example uses a PUT request to create a schedule that fires a script (`script/testlog.js`) every second. The example assumes that the script exists in the specified location. The schedule configuration is as described in Configure Schedules:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0" \
--request PUT \
--data '{
    "enabled": true,
    "type": "cron",
    "schedule": "0/1 * * * * ?",
    "persisted": true,
    "misfirePolicy": "fireAndProceed",
    "invokeService": "script",
    "invokeContext": {
        "script": {
            "type": "text/javascript",
            "file": "script/testlog.js"
        }
    }
}' \
"http://localhost:8080/openidm/scheduler/job/testlog-schedule"
{
  "_id": "testlog-schedule",
  "enabled": true,
  "persisted": true,
  "recoverable": false,
  "misfirePolicy": "fireAndProceed",
  "schedule": "0/1 * * * * ?",
  "repeatInterval": 0,
  "repeatCount": 0,
  "type": "cron",
  "invokeService": "org.forgerock.openidm.script",
  "invokeContext": {
    "script": {
      "type": "text/javascript",
      "file": "script/testlog.js"
    }
  },
  "invokeLogLevel": "info",
  "startTime": null,
  "endTime": null,
  "concurrentExecution": false,
  "triggers": [
    {
      "calendar": null,
      "group": "scheduler-service-group",
      "jobKey": "scheduler-service-group.testlog-schedule",
      "name": "trigger-testlog-schedule",
      "nodeId": "node1",
      "previousState": null,
      "serialized": {
        "type": "CronTriggerImpl",
        "calendarName": null,
        "cronEx": {
```

```
      "cronExpression": "0/1 * * * * ?",
      "timeZone": "Africa/Johannesburg"
    },
    "description": null,
    "endTime": null,
    "fireInstanceId": "node1_1570611359345",
    "group": "scheduler-service-group",
    "jobDataMap": {
      "scheduler.invokeService": "org.forgerock.openidm.script",
      "scheduler.config-name": "scheduler-testlog-schedule",
      "scheduler.invokeContext": {
        "script": {
          "type": "text/javascript",
          "file": "script/testlog.js"
        }
      },
      "schedule.config": {
        "enabled": true,
        "persisted": true,
        "recoverable": false,
        "misfirePolicy": "fireAndProceed",
        "schedule": "0/1 * * * * ?",
        "repeatInterval": 0,
        "repeatCount": 0,
        "type": "cron",
        "invokeService": "org.forgerock.openidm.script",
        "invokeContext": {
          "script": {
            "type": "text/javascript",
            "file": "script/testlog.js"
          }
        },
        "invokeLogLevel": "info",
        "startTime": null,
        "endTime": null,
        "concurrentExecution": false
      },
      "scheduler.invokeLogLevel": "info"
    },
    "jobGroup": "scheduler-service-group",
    "jobName": "testlog-schedule",
    "misfireInstruction": 1,
    "name": "trigger-testlog-schedule",
    "nextFireTime": 1570611569000,
    "previousFireTime": 1570611568000,
    "priority": 5,
    "startTime": 1570611391000,
    "volatility": false
  },
  "state": "NORMAL",
  "_rev": "000000001d4724d6",
  "_id": "scheduler-service-group.trigger-testlog-schedule"
```

```
    }
  ],
  "previousRunDate": "2019-10-09T08:59:28.000Z",
  "nextRunDate": "2019-10-09T08:59:29.000Z"
}
```

Note

The previous output includes the `trigger` that was created as part of the scheduled job, as well as the `nextRunDate` for the job. For more information about the `trigger` properties, refer to Query Schedule Triggers.

The following example uses a POST request to create an identical schedule to the one created in the previous example, but with a *server-assigned ID*:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0" \
--request POST \
--data '{
    "enabled": true,
    "type": "cron",
    "schedule": "0/1 * * * * ?",
    "persisted": true,
    "misfirePolicy": "fireAndProceed",
    "invokeService": "script",
    "invokeContext": {
        "script": {
            "type": "text/javascript",
            "file": "script/testlog.js"
        }
    }
}' \
"http://localhost:8080/openidm/scheduler/job?_action=create"
{
  "_id": "b12e4a77-a626-4a38-a1dc-8edc7498ca1c",
  "enabled": true,
  "persisted": true,
  "recoverable": false,
  "misfirePolicy": "fireAndProceed",
  "schedule": "0/1 * * * * ?",
  "repeatInterval": 0,
  "repeatCount": 0,
  "type": "cron",
  "invokeService": "org.forgerock.openidm.script",
  "invokeContext": {
    "script": {
      "type": "text/javascript",
      "file": "script/testlog.js"
    }
  },
  "invokeLogLevel": "info",
  "startTime": null,
  "endTime": null,
  "concurrentExecution": false,
  "triggers": [
    {
      "calendar": null,
      "group": "scheduler-service-group",
      "jobKey": "scheduler-service-group.b12e4a77-a626-4a38-a1dc-8edc7498ca1c",
      "name": "trigger-b12e4a77-a626-4a38-a1dc-8edc7498ca1c",
      "nodeId": null,
      "previousState": null,
      "serialized": {
        "type": "CronTriggerImpl",
        "calendarName": null,
        "cronEx": {
```

```
      "cronExpression": "0/1 * * * * ?",
      "timeZone": "Africa/Johannesburg"
    },
    "description": null,
    "endTime": null,
    "fireInstanceId": null,
    "group": "scheduler-service-group",
    "jobDataMap": {
      "scheduler.invokeService": "org.forgerock.openidm.script",
      "scheduler.config-name": "scheduler-b12e4a77-a626-4a38-a1dc-8edc7498ca1c",
      "scheduler.invokeContext": {
        "script": {
          "type": "text/javascript",
          "file": "script/testlog.js"
        }
      },
      "schedule.config": {
        "enabled": true,
        "persisted": true,
        "recoverable": false,
        "misfirePolicy": "fireAndProceed",
        "schedule": "0/1 * * * * ?",
        "repeatInterval": 0,
        "repeatCount": 0,
        "type": "cron",
        "invokeService": "org.forgerock.openidm.script",
        "invokeContext": {
          "script": {
            "type": "text/javascript",
            "file": "script/testlog.js"
          }
        },
        "invokeLogLevel": "info",
        "startTime": null,
        "endTime": null,
        "concurrentExecution": false
      },
      "scheduler.invokeLogLevel": "info"
    },
    "jobGroup": "scheduler-service-group",
    "jobName": "b12e4a77-a626-4a38-a1dc-8edc7498ca1c",
    "misfireInstruction": 1,
    "name": "trigger-b12e4a77-a626-4a38-a1dc-8edc7498ca1c",
    "nextFireTime": 1570611659000,
    "previousFireTime": null,
    "priority": 5,
    "startTime": 1570611659000,
    "volatility": false
  },
  "state": "NORMAL",
  "_rev": "000000009e2e2212",
  "_id": "scheduler-service-group.trigger-b12e4a77-a626-4a38-a1dc-8edc7498ca1c"
```

```
    }
  ],
  "previousRunDate": null,
  "nextRunDate": "2019-10-09T09:00:59.000Z"
}
```

The output includes the generated `_id` of the schedule, in this case:

```
"_id": "b12e4a77-a626-4a38-a1dc-8edc7498ca1c"
```

View scheduled job details

The following example displays the details of the schedule created in the previous example. Specify the job ID in the URL:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request GET \
"http://localhost:8080/openidm/scheduler/job/testlog-schedule"
{
  "_id": "testlog-schedule",
  "enabled": true,
  "persisted": true,
  "recoverable": false,
  "misfirePolicy": "fireAndProceed",
  "schedule": "0/1 * * * * ?",
  "repeatInterval": 0,
  "repeatCount": 0,
  "type": "cron",
  "invokeService": "org.forgerock.openidm.script",
  "invokeContext": {
    "script": {
      "type": "text/javascript",
      "file": "script/testlog.js"
    }
  },
  "invokeLogLevel": "info",
  "startTime": null,
  "endTime": null,
  "concurrentExecution": false,
  "triggers": [
    {
      "calendar": null,
      "group": "scheduler-service-group",
      "jobKey": "scheduler-service-group.testlog-schedule",
      "name": "trigger-testlog-schedule",
      "nodeId": null,
      "previousState": null,
      "serialized": {
        "type": "CronTriggerImpl",
        "calendarName": null,
        "cronEx": {
          "cronExpression": "0/1 * * * * ?",
          "timeZone": "Africa/Johannesburg"
        },
        "description": null,
        "endTime": null,
        "fireInstanceId": "node1_1570611359712",
        "group": "scheduler-service-group",
        "jobDataMap": {
          "scheduler.invokeService": "org.forgerock.openidm.script",
          "scheduler.config-name": "scheduler-testlog-schedule",
          "scheduler.invokeContext": {
            "script": {
              "type": "text/javascript",
              "file": "script/testlog.js"
            }
```

```
          },
          "schedule.config": {
            "enabled": true,
            "persisted": true,
            "recoverable": false,
            "misfirePolicy": "fireAndProceed",
            "schedule": "0/1 * * * * ?",
            "repeatInterval": 0,
            "repeatCount": 0,
            "type": "cron",
            "invokeService": "org.forgerock.openidm.script",
            "invokeContext": {
              "script": {
                "type": "text/javascript",
                "file": "script/testlog.js"
              }
            },
            "invokeLogLevel": "info",
            "startTime": null,
            "endTime": null,
            "concurrentExecution": false
          },
          "scheduler.invokeLogLevel": "info"
        },
        "jobGroup": "scheduler-service-group",
        "jobName": "testlog-schedule",
        "misfireInstruction": 1,
        "name": "trigger-testlog-schedule",
        "nextFireTime": 1570611719000,
        "previousFireTime": 1570611718000,
        "priority": 5,
        "startTime": 1570611391000,
        "volatility": false
      },
      "state": "NORMAL",
      "_rev": "000000002d1c2465",
      "_id": "scheduler-service-group.trigger-testlog-schedule"
    }
  ],
  "previousRunDate": "2019-10-09T09:01:58.000Z",
  "nextRunDate": "2019-10-09T09:01:59.000Z"
}
```


Query scheduled jobs

You can query defined and running scheduled jobs using a regular query filter.

The following query returns the IDs of all defined schedules:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request GET \
"http://localhost:8080/openidm/scheduler/job?_queryFilter=true&_fields=_id"
{
  "result": [
    {
      "_id": "reconcile_systemLdapAccounts_managedUser"
    },
    {
      "_id": "testlog-schedule"
    }
  ]
  ...
}
```

The following query returns the IDs, enabled status, and next run date of all defined schedules:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request GET \
"http://localhost:8080/openidm/scheduler/job?_queryFilter=true&_fields=_id,enabled,nextRunDate"
{
  "result": [
    {
      "_id": "reconcile_systemLdapAccounts_managedUser",
      "enabled": false,
      "nextRunDate": null
    },
    {
      "_id": "testlog-schedule",
      "enabled": true,
      "nextRunDate": "2019-10-09T09:43:17.000Z"
    }
  ]
  ...
}
```

Update a schedule

To update a schedule definition, use a PUT request and update all the static properties of the object.

This example disables the `testlog` schedule created in the previous example by setting `"enabled":false`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0" \
--request PUT \
--data '{
    "enabled": false,
    "type": "cron",
    "schedule": "0/1 * * * * ?",
    "persisted": true,
    "misfirePolicy": "fireAndProceed",
    "invokeService": "script",
    "invokeContext": {
        "script": {
            "type": "text/javascript",
            "file": "script/testlog.js"
        }
    }
}' \
"http://localhost:8080/openidm/scheduler/job/testlog-schedule"
{
  "_id": "testlog-schedule",
  "enabled": false,
  "persisted": true,
  "recoverable": false,
  "misfirePolicy": "fireAndProceed",
  "schedule": "0/1 * * * * ?",
  "repeatInterval": 0,
  "repeatCount": 0,
  "type": "cron",
  "invokeService": "org.forgerock.openidm.script",
  "invokeContext": {
    "script": {
      "type": "text/javascript",
      "file": "script/testlog.js"
    }
  },
  "invokeLogLevel": "info",
  "startTime": null,
  "endTime": null,
  "concurrentExecution": false,
  "triggers": [],
  "previousRunDate": null,
  "nextRunDate": null
}
```

When you disable a schedule, all triggers are removed, and the `nextRunDate` is set to `null`. If you re-enable the schedule, a new trigger is generated, and the `nextRunDate` is recalculated.

List running scheduled jobs

This example returns a list of the jobs that are currently executing. The list lets you decide whether to wait for a specific job to complete before shutting down a server.

> **Note**
>
> - This action does not list the jobs across a cluster, only the jobs currently executing on the node to which the request is routed.
> - The list is accurate only at the moment the request was issued, and can change at any time after the response is received.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request POST \
"http://localhost:8080/openidm/scheduler/job?_action=listCurrentlyExecutingJobs"
[
  {
    "enabled": true,
    "persisted": true,
    "misfirePolicy": "fireAndProceed",
    "type": "simple",
    "repeatInterval": 3600000,
    "repeatCount": -1,
    "invokeService": "org.forgerock.openidm.sync",
    "invokeContext": {
      "action": "reconcile",
      "mapping": "systemLdapAccounts_managedUser"
    },
    "invokeLogLevel": "info",
    "timeZone": null,
    "startTime": null,
    "endTime": null,
    "concurrentExecution": false
  }
]
```

Trigger a schedule manually

For testing purposes, and for certain administrative tasks, you can trigger a scheduled task manually, outside of its specified schedule. A scheduled task must be `enabled` before it can be triggered.

This command triggers the `testlog-schedule` job created previously:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request POST \
"http://localhost:8080/openidm/scheduler/job/testlog-schedule?_action=trigger"
{
  "success": true
}
```

> **Note**
>
> This action is available only from version 2.0 of the scheduler API onwards.

Pause and resume a scheduled job

Instead of deleting and recreating scheduled jobs, you can pause and resume them if necessary. This command pauses the `testlog-schedule` job:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request POST \
"http://localhost:8080/openidm/scheduler/job/testlog-schedule?_action=pause"
{
  "success": true
}
```

This command resumes the `testlog-schedule` job:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request POST \
"http://localhost:8080/openidm/scheduler/job/testlog-schedule?_action=resume"
{
  "success": true
}
```

> **Note**
>
> These actions are available only from version 2.0 of the scheduler API onwards.

Pause all scheduled jobs

You can temporarily suspend all scheduled jobs. This action does not cancel or interrupt jobs that are already in progress; it simply prevents any scheduled jobs from being invoked during the suspension period.

This command suspends all scheduled tasks and returns `true` if the tasks could be suspended successfully:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request POST \
"http://localhost:8080/openidm/scheduler/job?_action=pauseJobs"
{
  "success": true
}
```

Resume all scheduled jobs

You can resume scheduled jobs to start them up again. Any jobs that were missed during the downtime follow their configured `misfirePolicy`.

This command resumes all scheduled jobs and returns `true` if the jobs could be resumed successfully:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request POST \
"http://localhost:8080/openidm/scheduler/job?_action=resumeJobs"
{
  "success": true
}
```

Query schedule triggers

When a scheduled job is created, a trigger for that job is created automatically and is included in the schedule definition. The trigger is essentially what causes the job to be started. You can read all the triggers that have been generated on a system with the following query on the `openidm/scheduler/trigger` context path:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request GET \
"http://localhost:8080/openidm/scheduler/trigger?_queryFilter=true"
{
  "result": [
    {
      "_id": "scheduler-service-group.trigger-testlog-schedule",
      "_rev": "00000000db3523f1",
      "calendar": null,
      "group": "scheduler-service-group",
      "jobKey": "scheduler-service-group.testlog-schedule",
      "name": "trigger-testlog-schedule",
      "nodeId": "node1",
      "previousState": null,
      "serialized": {
      ...
      },
      "state": "NORMAL"
    }
  ]
}
```

The trigger object contents are:

`_id`

The ID of the trigger, which is based on the schedule ID. The trigger ID is made up of the group name, followed by `trigger-` prepended to the schedule ID: `group.trigger-schedule-id`. For example, if the schedule ID was `testlog-schedule`, then the trigger ID would be `scheduler-service-group.trigger-testlog-schedule`.

`_rev`

The revision of the trigger object. This property is reserved for internal use and specifies the revision of the object in the repository. This is the same value that is exposed as the object's ETag through the REST API. The content of this property is not defined. No consumer should make any assumptions of its content beyond equivalence comparison.

`previousState`

The previous state of the trigger, before its current state. For a description of Quartz trigger states, refer to the Quartz API documentation.

`name`

The trigger name, which matches the ID of the schedule that created the trigger, with `trigger-` added: `trigger-schedule-id`.

`state`

The current state of the trigger. For a description of Quartz trigger states, refer to the Quartz API documentation.

`nodeId`

The ID of the node that has acquired the trigger, useful in a clustered deployment. If the trigger has not been acquired by a node yet, this will return `null`.

`calendar`

This is a part of the Quartz implementation, but is not currently supported by IDM. This will always return `null`.

`serialized`

The JSON serialization of the trigger class.

`group`

The name of the group that the trigger is in, always `scheduler-service-group`.

`jobKey`

The name of the job associated with the trigger: `group.schedule-id`.

To read the contents of a specific trigger, send a GET request to the trigger ID; for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request GET \
"http://localhost:8080/openidm/scheduler/trigger/scheduler-service-group.trigger-testlog-schedule"
{
  "_id": "scheduler-service-group.trigger-testlog-schedule",
  "_rev": "00000000cd1723dd",
  "calendar": null,
  "group": "scheduler-service-group",
  "jobKey": "scheduler-service-group.testlog-schedule",
  "name": "trigger-testlog-schedule",
  "nodeId": "node1",
  "previousState": null,
  "serialized": {
  ...
  },
  "state": "NORMAL"
}
```

To view the triggers that have been acquired, send a GET request to the scheduler, with a `_queryFilter` of `nodeId`. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request GET \
"http://localhost:8080/openidm/scheduler/trigger?_queryFilter=(nodeId+pr)"
```

To view the triggers that have not yet been acquired by any node, send a GET request to the scheduler, with a `_queryFilter` to list the triggers with a null `nodeId`. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request GET \
"http://localhost:8080/openidm/scheduler/trigger?_queryFilter=%21(nodeId+pr)"
```

Delete a schedule

To delete a schedule, send a DELETE request to the schedule ID. For example:

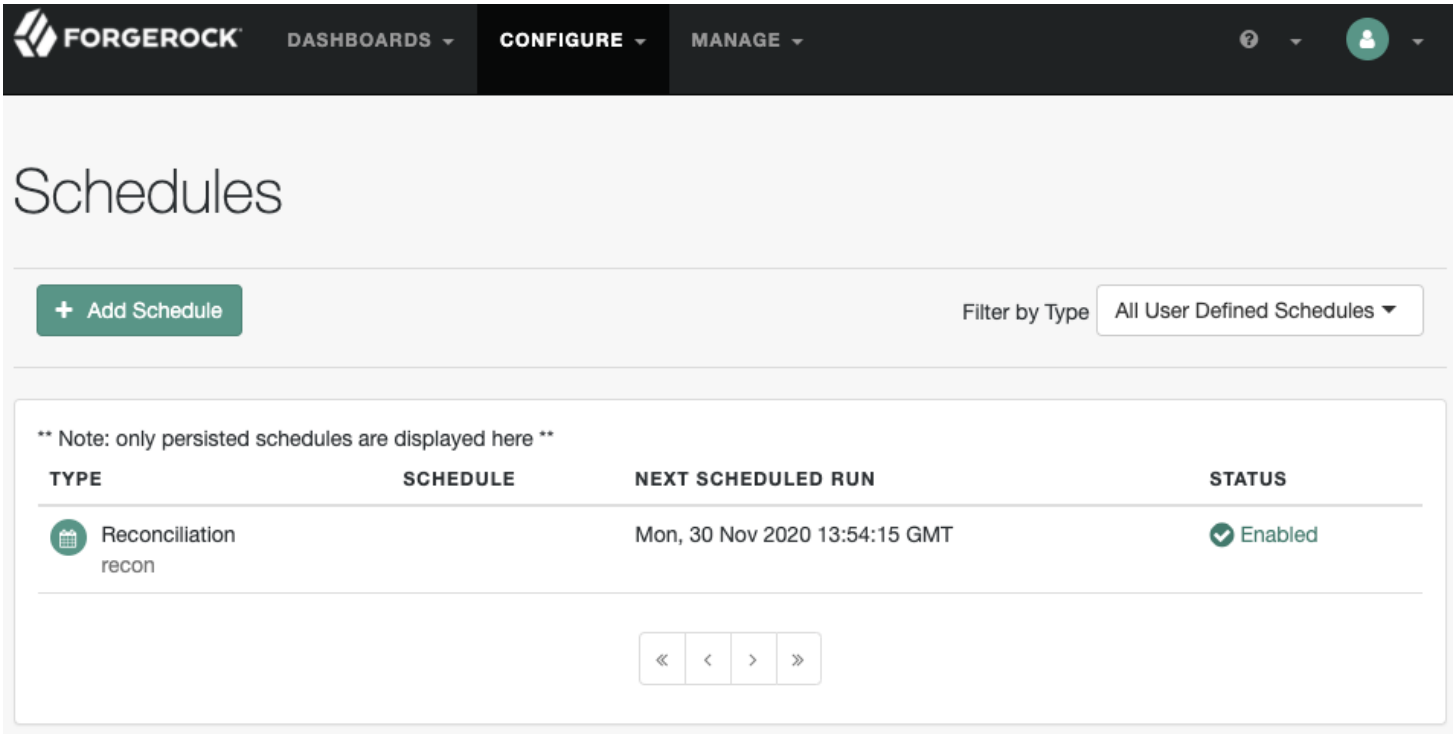
```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=2.0" \
--request DELETE \
"http://localhost:8080/openidm/scheduler/job/testlog-schedule"
{
  "_id": "testlog-schedule",
  "enabled": false,
  "persisted": true,
  "recoverable": false,
  "misfirePolicy": "fireAndProceed",
  "schedule": "0/1 * * * * ?",
  "repeatInterval": 0,
  "repeatCount": 0,
  "type": "cron",
  "invokeService": "org.forgerock.openidm.script",
  "invokeContext": {
    "script": {
      "type": "text/javascript",
      "file": "script/testlog.js"
    }
  },
  "invokeLogLevel": "info",
  "startTime": null,
  "endTime": null,
  "concurrentExecution": false,
  "triggers": [],
  "previousRunDate": null,
  "nextRunDate": null
}
```

The DELETE request returns the entire JSON object.

Manage schedules using the admin UI

To manage schedules using the admin UI, click **Configure > Schedules**.

Add, remove, and change schedules here. By default, only persisted schedules are shown in the **Schedules** list. To show non-persisted (in memory) schedules, select **Filter by Type > In Memory**.



Schedules and daylight savings time

The scheduler service supports Quartz cron triggers and simple triggers. Cron triggers schedule jobs to fire at specific times with respect to a calendar (rather than every N milliseconds). This scheduling can cause issues when clocks change for daylight savings time (DST) if the trigger time falls around the clock change time in your specific time zone.

Depending on the trigger schedule, and on the daylight event, the trigger might be skipped or might appear not to fire for a short period. This interruption can be particularly problematic for liveSync where schedules execute continuously. In this case, the time change (for example, from 02:00 back to 01:00) causes an hour break between each liveSync execution.

To prevent DST from having an impact on your schedules, use simple triggers instead of cron triggers.

Persistent schedules

By default, scheduling information (such as schedule state, and the details of the schedule run) is stored in RAM. This means that such information is lost when the server is rebooted. The schedule configuration is not lost when the server is shut down, and normal scheduling continues when the server is restarted. However, there are no details of missed schedule runs that should have occurred during the period the server was unavailable.

You can configure schedules to be persistent, which means that the scheduling information is stored in the internal repository, rather than in RAM. With persistent schedules, scheduling information is retained when the server is shut down. Any previously scheduled jobs can be rescheduled automatically when the server is restarted.

Persistent schedules also let you manage scheduling across a cluster (multiple IDM instances). When scheduling is persistent, a particular schedule will be launched only once across the cluster, rather than once on every instance. For example, if your deployment includes a cluster of nodes for high availability, you can use persistent scheduling to start a reconciliation operation on only one node in the cluster, instead of starting several competing reconciliation operations on each node.

> **Important**
>
> Persistent schedules rely on timestamps. In a deployment where IDM instances run on separate machines, you *must* synchronize the system clocks of these machines using a time synchronization service that runs regularly. The clocks of all machines involved in persistent scheduling must be within one second of each other. For information on how you can achieve this using the Network Time Protocol (NTP) daemon, refer to the NTP RFC.

To configure persistent schedules, set `persisted` to `true` in the schedule configuration.

If the server is down when a scheduled task was set to occur, one or more runs of that schedule might be missed. To specify what action should be taken if schedules are missed, set the `misfirePolicy` in the schedule configuration file. The `misfirePolicy` determines what IDM should do if scheduled tasks are missed. Possible values are as follows:

`fireAndProceed`

The first run of a missed schedule is immediately implemented when the server is back online. Subsequent runs are discarded. After this, the normal schedule is resumed.

`doNothing`

All missed schedules are discarded and the normal schedule is resumed when the server is back online.

Schedule examples

The following example shows a schedule for reconciliation that is not enabled. When the schedule is enabled (`"enabled" : true,`), reconciliation runs every 30 minutes (1800000 milliseconds), and repeats indefinitely:

```
{
    "enabled": false,
    "persisted": true,
    "type": "simple",
    "repeatInterval": 1800000,
    "invokeService": "sync",
    "invokeContext": {
        "action": "reconcile",
        "mapping": "systemLdapAccounts_managedUser"
    }
}
```

The following example shows a schedule for liveSync enabled to run every 15 seconds, repeating indefinitely. Note that the schedule is persisted; that is, stored in the repository rather than in memory. If one or more liveSync runs are missed, as a result of the server being unavailable, the first run of the liveSync operation is implemented when the server is back online. Subsequent runs are discarded. After this, the normal schedule is resumed:

```
{
    "enabled": true,
    "persisted": true,
    "misfirePolicy" : "fireAndProceed",
    "type": "simple",
    "repeatInterval": 15000,
    "invokeService": "provisioner",
    "invokeContext": {
        "action": "liveSync",
        "source": "system/ldap/account"
    }
}
```

Scan data to trigger tasks

In addition to the fine-grained scheduling facility, IDM provides a task scanning mechanism. The task scanner lets you scan a set of properties with a complex query filter, at a scheduled interval, and then launches a script on the objects returned by the query.

For example, the task scanner can scan all `managed/user` objects for a specific date, and invoke a script that launches a task on the user object when that date is reached.

The task scanner runs a scheduled task that queries a managed object, then launches a script based on the query results. Scanning tasks are configured in the same way as standard scheduled tasks, as part of the schedule configuration, with the `invokeService` parameter set to `taskscanner`. The `invokeContext` parameter defines the scan details, and the task that should be launched when the specified condition is triggered.

Activate and deactivate accounts

The default IDM configuration includes two scanning tasks that *activate* and *deactivate* a user's `accountStatus`, based on their `activeDate` and `inactiveDate`. The tasks run once a day by default.

> **Note**
>
> Both tasks are disabled by default. To enable them, set `"enabled" : true` in the schedule configuration for each task.

The `activate` task

The `activate` task (`conf/schedule-taskscan_activate.json`) has the following configuration:

```
{
  "enabled" : false,
  "type" : "simple",
  "repeatInterval" : 86400000,
  "persisted" : true,
  "concurrentExecution" : false,
  "invokeService" : "taskscanner",
  "invokeContext" : {
    "waitForCompletion" : false,
    "numberOfThreads" : 5,
    "scan" : {
      "_queryFilter" : "((/activeDate le \"${Time.nowWithOffset}\") AND (!(/inactiveDate pr) or /inactiveDate ge \"$
{Time.nowWithOffset}\"))",
      "object" : "managed/user",
      "taskState" : {
        "started" : "/activateAccount/task-started",
        "completed" : "/activateAccount/task-completed"
      },
      "recovery" : {
        "timeout" : "10m"
      }
    },
    "task" : {
      "script" : {
        "type" : "text/javascript",
        "globals" : { },
        "source" : "var patch = [{ \"operation\" : \"replace\", \"field\" : \"/accountStatus\", \"value\" :
\"active\" }];\n\nlogger.debug(\"Performing Activate Account Task on {} ({})\", input.mail, objectID);
\n\nopenidm.patch(objectID, null, patch); true;"
      }
    }
  }
}
```

When you run this task, a user account is *activated* if both of the following are true:

- Their `activeDate` is less than or equal to the value of `Time.nowWithOffset`.
- Their `inactiveDate` is greater than or equal to the value of `Time.nowWithOffset`, or they do not have an `inactiveDate` set.

> **Note**
>
> `Time.nowWithOffset` is the current time plus the UTC time offset for the user's geographical region.

The `expire` task

The `expire` task (`conf/schedule-taskscan_expire.json`) has the following configuration:

```
{
  "enabled" : false,
  "type" : "simple",
  "repeatInterval" : 86400000,
  "persisted" : true,
  "concurrentExecution" : false,
  "invokeService" : "taskscanner",
  "invokeContext" : {
    "waitForCompletion" : false,
    "numberOfThreads" : 5,
    "scan" : {
      "_queryFilter" : "((/inactiveDate lt \"${Time.nowWithOffset}\") AND (!(/activeDate pr) or /activeDate le \"$
{Time.nowWithOffset}\"))",
      "object" : "managed/user",
      "taskState" : {
        "started" : "/expireAccount/task-started",
        "completed" : "/expireAccount/task-completed"
      },
      "recovery" : {
        "timeout" : "10m"
      }
    },
    "task" : {
      "script" : {
        "type" : "text/javascript",
        "globals" : { },
        "source" : "var patch = [{ \"operation\" : \"replace\", \"field\" : \"/accountStatus\", \"value\" :
\"inactive\" }];\n\nlogger.debug(\"Performing Expire Account Task on {} ({})\", input.mail, objectID);
\n\nopenidm.patch(objectID, null, patch); true;"
      }
    }
  }
}
```

When you run this task, a user account is *deactivated* if both of the following are true:

- Their `inactiveDate` (expiry date) is less than the value of `Time.nowWithOffset`.
- Their `activeDate` is less than or equal to the value of `Time.nowWithOffset`, or they do not have an `activeDate` set.

> **Note**
>
> `Time.nowWithOffset` is the current time plus the UTC time offset for the user's geographical region.

Create a new scanning task

The following example (`openidm/samples/example-configurations/task-scanner/conf/schedule-taskscan_sunset.json`) defines a scheduled scanning task that triggers a sunset script:

```
{
  "enabled" : true,
  "type" : "simple",
  "repeatInterval" : 3600000,
  "persisted": true,
  "concurrentExecution" : false,
  "invokeService" : "taskscanner",
  "invokeContext" : {
    "waitForCompletion" : false,
    "numberOfThreads" : 5,
    "scan" : {
      "_queryFilter" : "((/sunset/date lt \"${Time.now}\") AND !(/sunset/task-completed pr))",
      "object" : "managed/user",
      "taskState" : {
        "started" : "/sunset/task-started",
        "completed" : "/sunset/task-completed"
      },
      "recovery" : {
        "timeout" : "10m"
      }
    },
    "task" : {
      "script" : {
        "type" : "text/javascript",
        "file" : "script/sunset.js"
      }
    }
  }
}
```

The schedule configuration calls a script (`sunset.js`). To test the sample, copy this file to your project's `script` directory or replace `"file" : "script/sunset.js"` with the script source (`"source" : "contents of sunset.js"`). The sample script marks all user objects that match the specified conditions as inactive. You can use this sample script to trigger a specific workflow, or any other task associated with the sunset process.

The task will only execute on users who have a valid `sunset/date` field. You can add a `sunset/date` field to user entries over REST. To make the field visible in the admin UI, you must add it to your managed object configuration.

This example command adds a `sunset/date` field to `bjensen` 's entry, over REST:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '[{
  "operation" : "add",
  "field" : "sunset/date",
  "value" : "2019-12-20T12:00:00Z"
}]' \
"http://localhost:8080/openidm/managed/user?_action=patch&_queryFilter=userName+eq+'bjensen'"
```

The remaining properties in the schedule configuration are as follows:

The `invokeContext` parameter takes the following properties:

`waitForCompletion` *(optional)*

This property specifies whether the task should be performed synchronously. Tasks are performed asynchronously by default (with `waitForCompletion` set to false). A task ID (such as `{"_id":"354ec41f-c781-4b61-85ac-93c28c180e46"}`) is returned immediately. If this property is set to true, tasks are performed synchronously, and the ID is not returned until all tasks have completed.

`maxRecords` *(optional)*

The maximum number of records that can be processed. This property is not set by default so the number of records is unlimited. If a maximum number of records is specified, that number will be spread evenly over the number of threads.

`numberOfThreads` *(optional)*

By default, the task scanner runs in a multi-threaded manner; that is, numerous threads are dedicated to the same scanning task run. Multi-threading generally improves the performance of the task scanner. The default number of threads for a single scanning task is 10. To change this default, set the `numberOfThreads` property. The sample configuration sets the default number of threads to 5.

`scan`

The details of the scan. The following properties are defined:

`_queryFilter`

The query filter that identifies the entries for which this task should be run.

The query filter provided in the sample schedule configuration (`/sunset/date lt \"${Time.now}\") AND !(/sunset/task-completed pr`) identifies managed users whose `sunset/date` property is before the current date and for whom the sunset task has not yet completed.

The sample query supports time-based conditions, with the time specified in ISO 8601 format (Zulu time). You can write any query to target the set of entries that you want to scan.

For time-based queries, it's possible to use the `${Time.now}` macro object (which fetches the current time). You can also specify any date/time in relation to the current time, using the `` ` or `-` operator, and a duration modifier. For example: changing the sample query to `${Time.now + 1d}` would return all user objects whose `/sunset/date` is the following day (current time plus one day). Note: you must include space characters around the operator (`` or -`). The duration modifier supports the following unit specifiers:

- `s` second
- `m` minute
- `h` hour
- `d` day
- `M` month
- `y` year

`object`

Defines the managed object type against which the query should be performed, as defined in the `managed.json` file.

`taskState`

Indicates the names of the fields in which the start message, and the completed message are stored. These fields are used to track the status of the task.

`started`

specifies the field that stores the timestamp for when the task begins.

`completed`

specifies the field that stores the timestamp for when the task completes its operation. The `completed` field is present as soon as the task has started, but its value is `null` until the task has completed.

`recovery` *(optional)*

Specifies a configurable `timeout`, after which the task scanner process ends. For clustered IDM instances, there might be more than one task scanner running at a time. A task cannot be launched by two task scanners at the same time. When one task scanner "claims" a task, it indicates that the task has been started. That task is then unavailable to be claimed by another task scanner and remains unavailable until the end of the task is indicated. In the event that the first task scanner does not complete the task by the specified timeout, for whatever reason, a second task scanner can pick up the task.

`task`

Provides details of the task that is performed. Usually, the task is invoked by a script, whose details are defined in the `script` property:

`type`

The script type.

IDM supports `"text/javascript"` and `"groovy"`.

`file`

The path to the script file. The script file takes at least two objects (in addition to the default objects that are provided to all IDM scripts):

`input`

The individual object that is retrieved from the query (in the example, this is the individual user object).

`objectID`

A string that contains the full identifier of the object. The `objectID` is useful for performing updates with the script as it allows you to target the object directly. For example: `openidm.update(objectID, input['_rev'], input);`.

> **Note**
>
> For more information about using scripts, refer to Scripting function reference.

Manage scanning tasks

Once you create scanning tasks, you may want to update them. For example, you might want to trigger, cancel, or list the existing scanning tasks.

You can manage scanning tasks in IDM using:

- REST
- Admin UI

Manage scanning tasks using REST

You can trigger, cancel, and monitor scanning tasks over the REST interface, using the REST endpoint `openidm/taskscanner`.

Create a scanning task

You can define a scanning task in a configuration file or directly over the REST interface. For an example of a file-based scanning task, refer to the file `/path/to/openidm/samples/example-configurations/task-scanner/conf/schedule-taskscan_sunset.json`.

The following command defines a scanning task named `sunsetTask`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-type: application/json" \
--header "Accept-API-Version: resource=2.0" \
--request PUT \
--data '{
  "enabled" : true,
  "type" : "simple",
  "repeatInterval" : 3600000,
  "persisted": true,
  "concurrentExecution" : false,
  "invokeService" : "taskscanner",
  "invokeContext" : {
    "waitForCompletion" : false,
    "numberOfThreads" : 5,
    "scan" : {
      "_queryFilter" : "/sunset/date lt \"$\{Time.now}\") AND !(/sunset/task-completed pr",
      "object" : "managed/user",
      "taskState" : {
        "started" : "/sunset/task-started",
        "completed" : "/sunset/task-completed"
      },
      "recovery" : {
        "timeout" : "10m"
      }
    },
    "task" : {
      "script" : {
        "type" : "text/javascript",
        "file" : "script/sunset.js"
      }
    }
  }
}' \
"http://localhost:8080/openidm/scheduler/job/sunsetTask"
{
  "_id": "sunsetTask",
  "enabled": true,
  "persisted": true,
  "recoverable": false,
  "misfirePolicy": "fireAndProceed",
  "schedule": null,
  "repeatInterval": 3600000,
  "repeatCount": -1,
  "type": "simple",
  "invokeService": "org.forgerock.openidm.taskscanner",
  "invokeContext": {
    "waitForCompletion": false,
    "numberOfThreads": 5,
    "scan": {
      "_queryFilter": "/sunset/date lt \"$\{Time.now}\") AND !(/sunset/task-completed pr",
      "object": "managed/user",
      "taskState": {
```

```
        "started": "/sunset/task-started",
        "completed": "/sunset/task-completed"
      },
      "recovery": {
        "timeout": "10m"
      }
    },
    "task": {
      "script": {
        "type": "text/javascript",
        "file": "script/sunset.js"
      }
    }
  },
  "invokeLogLevel": "info",
  "startTime": null,
  "endTime": null,
  "concurrentExecution": false,
  "triggers": [
    {
      "calendar": null,
      "group": "scheduler-service-group",
      "jobKey": "scheduler-service-group.sunsetTask",
      "name": "trigger-sunsetTask",
      "nodeId": null,
      "previousState": null,
      "serialized": {
        "type": "SimpleTriggerImpl",
        "calendarName": null,
        "complete": false,
        "description": null,
        "endTime": null,
        "fireInstanceId": null,
        "group": "scheduler-service-group",
        "jobDataMap": {
          "scheduler.invokeService": "org.forgerock.openidm.taskscanner",
          "scheduler.config-name": "scheduler-sunsetTask",
          "scheduler.invokeContext": {
            "waitForCompletion": false,
            "numberOfThreads": 5,
            "scan": {
              "_queryFilter": "/sunset/date lt \"$\{Time.now}\") AND !(/sunset/task-completed pr",
              "object": "managed/user",
              "taskState": {
                "started": "/sunset/task-started",
                "completed": "/sunset/task-completed"
              },
              "recovery": {
                "timeout": "10m"
              }
            },
            "task": {
              "script": {
                "type": "text/javascript",
                "file": "script/sunset.js"
```

```
            }
          }
        },
        "schedule.config": {
          "enabled": true,
          "persisted": true,
          "recoverable": false,
          "misfirePolicy": "fireAndProceed",
          "schedule": null,
          "repeatInterval": 3600000,
          "repeatCount": -1,
          "type": "simple",
          "invokeService": "org.forgerock.openidm.taskscanner",
          "invokeContext": {
            "waitForCompletion": false,
            "numberOfThreads": 5,
            "scan": {
              "_queryFilter": "/sunset/date lt \"$\{Time.now}\") AND !(/sunset/task-completed pr",
              "object": "managed/user",
              "taskState": {
                "started": "/sunset/task-started",
                "completed": "/sunset/task-completed"
              },
              "recovery": {
                "timeout": "10m"
              }
            },
            "task": {
              "script": {
                "type": "text/javascript",
                "file": "script/sunset.js"
              }
            }
          },
          "invokeLogLevel": "info",
          "startTime": null,
          "endTime": null,
          "concurrentExecution": false
        },
        "scheduler.invokeLogLevel": "info"
      },
      "jobGroup": "scheduler-service-group",
      "jobName": "sunsetTask",
      "misfireInstruction": 1,
      "name": "trigger-sunsetTask",
      "nextFireTime": 1570618094818,
      "previousFireTime": null,
      "priority": 5,
      "repeatCount": -1,
      "repeatInterval": 3600000,
      "startTime": 1570618094818,
      "timesTriggered": 0,
      "volatility": false
    },
    "state": "NORMAL",
```

```
      "_rev": "000000006751ccf1",
      "_id": "scheduler-service-group.trigger-sunsetTask"
    }
  ],
  "previousRunDate": null,
  "nextRunDate": "2019-10-09T10:48:14.818Z"
}
```

Trigger a scanning task

To trigger a scanning task over REST, use the `execute` action and specify the `name` of the task (effectively the scheduled job name). To obtain a list of task names, you can query the `/openidm/scheduler/job` endpoint. Note, however, that not all jobs are scanning tasks. Only those jobs that have which have the correct task scanner `invokeContext` can be triggered in this way.

The following example triggers the `sunsetTask` defined in the previous example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/taskscanner?_action=execute&name=sunsetTask"
{
  "_id": "9f2564c8-193c-4871-8869-6080f374b1bd-2073"
}
```

For scanning tasks that are defined in configuration files, you can determine the task name from the file name, for example, `schedule-task-name.json`. The following example triggers a task named `taskscan_sunset` that is defined in a file named `conf/schedule-taskscan_sunset.json`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/taskscanner?_action=execute&name=taskscan_sunset"
{
  "_id": "8d7742f0-5245-41cf-89a5-de32fc50e326-3323"
}
```

By default, a scanning task ID is returned immediately when the task is initiated. Clients can make subsequent calls to the task scanner service, using this task ID to query its state and to call operations on it.

To have the scanning task complete before the ID is returned, set the `waitForCompletion` property to `true` in the task definition file (`schedule-taskscan_sunset.json`).

Cancel a scanning task

To cancel a scanning task that is in progress, send a REST call with the `cancel` action, specifying the task ID. The following call cancels the scanning task initiated in the previous example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/taskscanner/9f2564c8-193c-4871-8869-6080f374b1bd-2073?_action=cancel"
{
  "_id":"9f2564c8-193c-4871-8869-6080f374b1bd-2073",
  "status":"SUCCESS"
}
```

Note

You cannot cancel a scanning task that has already completed.

List the scanning tasks

To retrieve a list of scanning tasks, query the `openidm/taskscanner` context path. The following example displays *all* scanning tasks, regardless of their state:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/taskscanner?_queryFilter=true"
{
  "result": [
    {
      "_id": "9f2564c8-193c-4871-8869-6080f374b1bd-2073",
      "name": "schedule/taskscan_sunset",
      "progress": {
        "state": "COMPLETED",
        "processed": 0,
        "total": 0,
        "successes": 0,
        "failures": 0
      },
      "started": "2017-12-19T11:45:53.433Z",
      "ended": "2017-12-19T11:45:53.438Z"
    },
    {
      "_id": "b32aafe5-b484-4d00-89ff-83554341f321-9970",
      "name": "schedule/taskscan_sunset",
      "progress": {
        "state": "ACTIVE",
        "processed": 80,
        "total": 980,
        "successes": 80,
        "failures": 0
      },
      "started": "2017-12-19T16:41:04.185Z",
      "ended": null
    }
  ]
  ...
}
```

Each scanning task has the following properties:

`_id`

The unique ID of that task instance.

`name`

The name of the scanning task, determined by the name of the schedule configuration file or over REST when the task is executed.

`started`

The time at which the scanning task started.

ended

The time at which the scanning task ended.

progress

The progress of the scanning task, summarized in the following fields:

`failures`	The number of records not able to be processed.
`successes`	The number of records processed successfully.
`total`	The total number of records.
`processed`	The number of processed records.
`state`	The current state of the task, `INITIALIZED`, `ACTIVE`, `COMPLETED`, `CANCELLED`, or `ERROR`.

The number of processed tasks whose details are retained is governed by the `openidm.taskscanner.maxcompletedruns` property in the `conf/system.properties` file. By default, the last 100 completed tasks are retained.

Manage scanning tasks using the admin UI

The task scanner queries a set of managed objects, then executes a script on the objects returned in the query result. The scanner then sets a field on a specific managed object property to indicate the state of the task. Before you start, you must set up this object type property on the managed user object.

In the example that follows, the task scanner queries managed user objects and returns objects whose `sunset` property holds a date that is prior to the current date. The scanner then sets the state of the task in the `task-completed` field of the user's `sunset` property.

1. Click **Configure > Schedules**, and click **Add Schedule**.
2. Enable the schedule, and set the times that the task should run.
3. Under **Perform Action**, select **Execute a script on objects returned by a query (Task Scanner)**.
4. Select the managed object on which the query should be run; in this case, `user`.
5. Build the query that will be run against the managed user objects.

   The following query returns all managed users whose `sunset` date is prior to the current date (`${Time.now}`) and for whom the `sunset` task has not already completed (`/sunset/task-completed pr`):

   ```
   ((/sunset/date lt \"${Time.now}\") AND !(/sunset/task-completed pr))
   ```

6. In the **Object Property** Field, enter the property whose values will determine the state of the task; in this case, `sunset`.
7. In the **Script** field, enter an inline script, or a path to the file containing the script that should be launched on the results of the query.

   The sample task scanner runs the following script on the managed users returned by the previous query:

```
var patch = [{ "operation" : "replace", "field" : "/active", "value" : false },{ "operation" : "replace",
"field" : "/accountStatus", "value" : "inactive" }];
openidm.patch(objectID, null, patch);
```

This script essentially deactivates the accounts of users returned by the query by setting the value of their `active` property to `false`.

8. Configure the advanced properties of the schedule described in Configure Schedules.

External services

Configure external email and external REST access.

Email

Configure outbound email from IDM.

External REST

Access external REST services.

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

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

Outbound email

The outbound email service sends email from IDM, using a script or the REST API.

You can edit the email service over REST at the `config/external.email` endpoint, or in the `external.email.json` file in your project's `conf` directory.

Important

IDM supports UTF-8 (non-ASCII/international) characters in email addresses, such as **zoë@example.com**. When sending emails to these type of addresses, the configured SMTP server must also support UTF-8.

Sample email configuration

This sample email configuration sets up the outbound email service:

```
{
    "host" : "smtp.gmail.com",
    "port" : 587,
    "debug" : false,
    "auth" : {
        "enable" : true,
        "username" : "xxxxxxxx",
        "password" : "xxxxxxxx"
    },
    "timeout" : 300000,
    "writetimeout" : 300000,
    "connectiontimeout" : 300000,
    "starttls" : {
        "enable" : true
    },
    "ssl" : {
        "enable" : false
    },
    "smtpProperties" : [
        "mail.smtp.ssl.protocols=TLSv1.2",
        "mail.smtps.ssl.protocols=TLSv1.2"
    ],
    "threadPoolSize" : 20
}
```

Configure outbound email

To configure the outbound email service using the admin UI, click **Configure > Email Settings**.

1. Edit the email configuration with the mail server details and account. For the complete list of configuration options, refer to External email configuration properties.
2. Submit the configuration over REST, for example:

> **Tip**
>
> You can also copy the file to your project's `conf/` directory.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
    "host" : "smtp.gmail.com",
    "port" : 587,
    "debug" : false,
    "auth" : {
        "enable" : true,
        "username" : "admin",
        "password" : "Passw0rd"
    },
    "from" : "admin@example.com",
    "timeout" : 300000,
    "writetimeout" : 300000,
    "connectiontimeout" : 300000,
    "starttls" : {
        "enable" : true
    },
    "ssl" : {
        "enable" : false
    },
    "smtpProperties" : [
        "mail.smtp.ssl.protocols=TLSv1.2",
        "mail.smtps.ssl.protocols=TLSv1.2"
    ],
    "threadPoolSize" : 20
}' \
"http://localhost:8080/openidm/config/external.email"
```

Note

IDM encrypts the password.

External email configuration properties

`host`

The host name or IP address of the SMTP server. This can be the `localhost`, if the mail server is on the same system as IDM.

`port`

SMTP server port number, such as 25, 465, or 587.

Note

Many SMTP servers require the use of a secure port such as 465 or 587. Many ISPs flag email from port 25 as spam.

`debug`

When set to `true`, this option outputs diagnostic messages from the JavaMail library. Debug mode can be useful if you are having difficulty configuring the external email endpoint with your mail server.

`auth`

The authentication details for the mail account from which emails will be sent.

- `enable`—indicates whether you need login credentials to connect to the SMTP server.

  Note

  If `"enable" : false,` , you can leave the entries for `"username"` and `"password"` empty:

  ```
  "enable" : false,
  "username" : "",
  "password" : ""
  ```

- `username`—the account used to connect to the SMTP server.
- `password`—the password used to connect to the SMTP server.

`starttls`

If `"enable" : true`, enables the use of the STARTTLS command (if supported by the server) to switch the connection to a TLS-protected connection before issuing any login commands. If the server does not support STARTTLS, the connection continues without the use of TLS.

`from` *(optional)*

Specifies a default **From:** address which displays when users receive emails from IDM.

Important

Although `from` is optional in the email configuration, the email service requires this property to send email. If you do not specify a `from` address in the email configuration, you must provide one in another way, for example:

- From an email template.
- As a parameter in the email service request (`from` or `_from`).

`ssl`

Set `"enable" : true` to use SSL to connect, and to use the SSL port by default.

`smtpProperties`

Specifies the SSL protocols that will be enabled for SSL connections. Protocols are specified as a whitespace-separated list. The default protocol is TLSv1.2.

`threadPoolSize` *(optional)*

Emails are sent in separate threads managed by a thread pool. This property sets the number of concurrent emails that can be handled at a specific time. The default thread pool size (if none is specified) is `20`.

`connectiontimeout` *(integer, optional)*

The socket connection timeout, in milliseconds. The default connection timeout (if none is specified) is `300000` milliseconds, or 5 minutes. A setting of 0 disables this timeout.

`timeout` *(integer, optional)*

The socket read timeout, in milliseconds. The default read timeout (if none is specified) is `300000` milliseconds, or 5 minutes. A setting of 0 disables this timeout.

`writetimeout` *(integer, optional)*

The socket write timeout, in milliseconds. The default write timeout (if none is specified) is `300000` milliseconds, or 5 minutes. A setting of 0 disables this timeout.

Send mail using REST

In a production environment, you typically send mail from a script. To test your configuration, you can use the REST API by sending an HTTP POST to `/openidm/external/email`. You pass the message parameters as part of the POST payload, URL encoding the content, as necessary.

The following example sends a test email using the REST API:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "from":"openidm@example.com",
  "to":"your_email@example.com",
  "subject":"Test",
  "body":"Test"}' \
"http://localhost:8080/openidm/external/email?_action=send"
{
  "status": "OK",
  "message": "Email sent"
}
```

By default, a response is returned only when the SMTP relay has completed. To return a response immediately, without waiting for the SMTP relay to finish, include the parameter `waitForCompletion=false` in the REST call. Use this option only if you do not need to verify that the email was accepted by the SMTP server. For example:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "from":"openidm@example.com",
  "to":"your_email@example.com",
  "subject":"Test",
  "body":"Test"}' \
"http://localhost:8080/openidm/external/email?_action=send&waitForCompletion=false"
{
  "status": "OK",
  "message": "Email submitted"
}
```

Mail templates

You can send an email template using the `sendTemplate` action. For example:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "templateName":"welcome",
  "to":"your_email@example.com",
  "cc":"alt_email@example.com",
  "bcc":"bigBoss_email@example.com"}' \
"http://localhost:8080/openidm/external/email?_action=sendTemplate"
{
  "status": "OK",
  "message": "Email sent"
}
```

Note

Email templates utilize Handlebar expressions to reference object data dynamically. For example, to reference the `userName` of an object:

```
{{object.userName}}
```

Send mail using a script

You can send email using the resource API functions, with the `external/email` context. For more information about these functions, refer to openidm.action. In the following example, `params` is an object that contains the POST parameters:

```
var params =  new Object();
params.from = "openidm@example.com";
params.to = "your_email@example.com";
params.cc = "bjensen@example.com,scarter@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", "send", params);
```

Mail templates

You can send an email template using the `sendTemplate` action. For example:

Example 1

```
var params =  new Object();
params.templateName = "welcome";
params.to = "your_email@example.com";
params.cc = "bjensen@example.com,scarter@example.com";
params.bcc = "bigBoss@example.com";

openidm.action("external/email", "sendTemplate", params);
```

Example 2

```
var params = new Object();
params.templateName = "myTemplate";
params.to = "hgale815@example.com";
params.object = { "givenName": newObject.givenName, "sn": newObject.sn, "mail": newObject.mail, "country":
newObject.country };

openidm.action("external/email", "sendTemplate", params);
```

Note

Email templates utilize Handlebar expressions to reference object data dynamically. For example, to reference the `userName` of an object:

```
{{object.userName}}
```

`external/email` POST parameters

IDM 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"`.

Email rate limiting

No rate limiting is applied to password reset emails, or any emails sent by the IDM server. This means that an attacker can potentially spam a known user account with an infinite number of emails, filling that user's inbox. In the case of password reset, the spam attack can obscure an actual password reset attempt.

In a production environment, you must configure email rate limiting through the network infrastructure in which IDM runs. Configure the network infrastructure to detect and prevent frequent repeated requests to publicly accessible web pages, such as the password reset page. You can also handle rate limiting within your email server.

Access External REST Services

The external REST service lets you access remote REST services at the `openidm/external/rest` context path or by specifying the `external/rest` resource in your scripts. Note that this service is not intended as a full connector to synchronize or reconcile identity data, but as a way to make dynamic HTTP calls as part of the IDM logic. For more declarative and encapsulated interaction with remote REST services, and for synchronization or reconciliation operations, use the scripted REST implementation of the Groovy connector.

An external REST call via a script might look something like the following:

```
openidm.action("external/rest", "call", params);
```

The `call` parameter specifies the action name to be used for this invocation, and is the standard method signature for the `openidm.action` method.

An external REST call over REST might look something like the following:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "url": "http://urlecho.appspot.com/echo?status=200&Content-
Type=application%2Fjson&body=%5B%7B%22key%22%3A%22value%22%7D%5D",
  "method": "GET"
}' \
"http://localhost:8080/openidm/external/rest?_action=call"
[
  {
    "key": "value"
  }
]
```

Configure the External REST Service

You can edit the external REST configuration over REST at the `config/external.rest` endpoint, or in an `external.rest.json` file in your project's `conf` directory.

The following sample external REST configuration sets up the external REST service:

Using REST

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-type: application/json" \
--request PUT \
--data '{
  "socketTimeout" : "10 s",
  "connectionTimeout" : "10 s",
  "reuseConnections" : true,
  "retryRequests" : true,
  "maxConnections" : 64,
  "tlsVersion" : "&{openidm.external.rest.tls.version}",
  "hostnameVerifier" : "&{openidm.external.rest.hostnameVerifier}",
  "proxy" : {
    "proxyUri" : "",
    "userName" : "",
    "password" : ""
  }
}' \
"http://localhost:8080/openidm/config/external.rest"
```

```
{
  "_id": "external.rest",
  "socketTimeout": "10 s",
  "connectionTimeout": "10 s",
  "reuseConnections": true,
  "retryRequests": true,
  "maxConnections": 64,
  "tlsVersion": "&{openidm.external.rest.tls.version}",
  "hostnameVerifier": "&{openidm.external.rest.hostnameVerifier}",
  "proxy": {
    "proxyUri": "",
    "userName": "",
    "password": ""
  }
}
```

Using the filesystem

Copy the config to the `external.rest.json` file in your project's `conf` directory:

```
{
    "socketTimeout" : "10 s",
    "connectionTimeout" : "10 s",
    "reuseConnections" : true,
    "retryRequests" : true,
    "maxConnections" : 64,
    "tlsVersion" : "&{openidm.external.rest.tls.version}",
    "hostnameVerifier" : "&{openidm.external.rest.hostnameVerifier}",
    "proxy" : {
        "proxyUri" : "",
        "userName" : "",
        "password" : ""
    }
}
```

External REST Configuration Properties

`socketTimeout` *(string)*

The TCP socket timeout, in seconds, when waiting for HTTP responses. The default timeout is 10 seconds.

`connectionTimeout` *(string)*

The TCP connection timeout for new HTTP connections, in seconds. The default timeout is 10 seconds.

`reuseConnections` *(boolean, true or false)*

Specifies whether HTTP connections should be kept alive and reused for additional requests. By default, connections will be reused if possible.

`retryRequests` *(boolean, true or false)*

Specifies whether requests should be retried if a failure is detected. By default requests will be retried.

`maxConnections` *(integer)*

The maximum number of connections that should be pooled by the HTTP client. At most `64` connections will be pooled by default.

`tlsVersion` *(string)*

The TLS version that should be used for connections.

By default, TLS connections made via the external REST service use TLS version 1.2. In some cases, you might need to specify a different TLS version, for example, if you are connecting to a legacy system that supports an old version of TLS that is not accommodated by the backward-compatibility mode of your Java client. If you need to specify that the external REST service use a different TLS version, uncomment the `openidm.external.rest.tls.version` property towards the end of the `resolver/boot.properties` file and set its value, for example:

```
openidm.external.rest.tls.version=TLSv1.3
```

Valid versions for this parameter include TLSv1.1, TLSv1.2, and TLSv1.3.

`hostnameVerifier` *(string)*

Specifies whether the external REST service should check that the hostname to which an SSL client has connected is allowed by the certificate that is presented by the server.

The property can take the following values:

- `STRICT` - hostnames are validated
- `ALLOW_ALL` - the external REST service does not attempt to match the URL hostname to the SSL certificate Common Name, as part of its validation process

By default, this property is set in the `resolver/boot.properties` file and the value in `conf/external.rest.json` references that setting. For testing purposes, the default setting in `boot.properties` is:

```
openidm.external.rest.hostnameVerifier=ALLOW_ALL
```

If you do not set this property (by removing it from the `boot.properties` file or the `conf/external.rest.json` file), the behavior is to validate hostnames (the equivalent of setting `"hostnameVerifier": "STRICT"`). In production environments, you *should* set this property to `STRICT`.

`proxy`

Lets you set a proxy server *specific* to the external REST service. If you set a `proxyUri` here, the system-wide proxy settings described in HTTP Clients are ignored. To configure a system-wide proxy, leave these `proxy` settings empty and configure the HTTP Client settings instead.

Invocation Parameters

The following parameters are passed in the resource API parameters map. These parameters can override the static configuration (if present) on a per-invocation basis.

`url`

The target URL to invoke, in string format.

`method`

The HTTP action to invoke, in string format.

Possible actions include `POST`, `GET`, `PUT`, `DELETE`, and `OPTIONS`.

`headers` *(optional)*

The HTTP headers to set, in a map format from string (header-name) to string (header-value). For example, `Accept-Language: en-US`.

`contentType` *(optional)*

The media type of the data that is sent, for example `"contentType" : "application/json"`. This parameter is applied only if no `Content-Type` header is included in the request. (If a `Content-Type` header is included, that header takes precedence over this `contentType` parameter.) If no `Content-Type` is provided (in the header or with this parameter), the default content type is `application/json; charset=utf-8`.

`body` *(optional)*

The body or resource representation to send (for PUT and POST operations), in string format.

`base64` *(boolean, optional)*

Indicates that the `body` is base64-encoded, and should be decoded prior to transmission.

`forceWrap` *(boolean, optional)*

Indicates that the response must be wrapped in the headers/body JSON message format, even if the response was JSON, and would otherwise have been passed through unchanged.

If you need to disambiguate between HTTP 20x response codes, you must invoke the external REST service with `forceWrap=true`. For failure cases, the HTTP status code is present within the wrapped response embedded in the exception detail, or through the resource exception itself. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "url": "http://urlecho.appspot.com/echo?status=203&Content-
Type=application%2Fjson&body=%5B%7B%22key%22%3A%22value%22%7D%5D",
  "method": "GET",
  "forceWrap": true}' \
"http://localhost:8080/openidm/external/rest?_action=call"
{
  "headers": {
    "Access-Control-Allow-Origin": [
      "*"
    ],
    "Cache-Control": [
      "max-age=3600"
    ],
    "Content-Length": [
      "17"
    ],
    "Content-Type": [
      "application/json"
    ],
    "Date": [
      "Fri, 17 Jul 2020 10:55:54 GMT"
    ],
    "Server": [
      "Google Frontend"
    ],
    "X-Cloud-Trace-Context": [
      "11e4441659a85832e47af219d6e657af"
    ]
  },
  "code": 203,
  "body": [
    {
      "key": "value"
    }
  ]
}
```

`authenticate`

The authentication type, and the details with which to authenticate.

IDM supports the following authentication types:

- `basic` authentication with a username and password, for example:

```
"authenticate" : {
    "type": "basic",
    "user" : "john",
    "password" : "Passw0rd"
}
```

- `bearer` authentication, with an OAuth token instead of a username and password, for example:

```
"authenticate" : {
    "type": "bearer",
    "token" : "ya29.iQDWKpn8AHy09p....."
}
```

If no `authenticate` parameter is specified, no authentication is used.

Support for Non-JSON Responses

The external REST service supports any arbitrary payload (currently in stringified format). If the response is anything other than JSON, a JSON message object is returned:

- For text-compatible (non-JSON) content, IDM returns a JSON object similar to the following:

```
{
    "headers": { "Content-Type": ["..."] },
    "body": "..."
}
```

- Content that is not text-compatible (such as JPEGs) is base64-encoded in the response `body` and returned as follows:

```
{
    "headers": { "Content-Type": ["..."] },
    "body": "...",
    "base64": true
}
```

> **Note**
>
> If the response format is JSON, the raw JSON response is returned. If you want to inspect the response headers, set `forceWrap` to `true` in your request. This setting returns a JSON message object with `headers` and `body`, similar to the object returned for text-compatible content.

Monitoring and metrics

Configure ForgeRock® Identity Management server logs and monitoring metrics.

Server Logs

Manage and read server logs.

Monitoring

Set up systems to monitor IDM.

Metrics

Monitoring metrics reference information.

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

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

Server logs

Server logging is not the same as auditing. Auditing logs activity on the IDM system, such as access, and synchronization. Server logging records information about the internal workings of IDM, like system messages, error reporting, service loading, or startup and shutdown messaging.

Configure server logging in your project's `conf/logging.properties` file. Changes to logging settings require a server restart before they take effect. Alternatively, use JMX via jconsole to change the logging settings. In this case, changes take effect without restarting the server.

Log message handlers

The way IDM logs messages is set in the `handlers` property in the `logging.properties` file. This property has the following value by default:

```
handlers=java.util.logging.FileHandler, java.util.logging.ConsoleHandler
```

The default handlers are:

- `FileHandler` writes formatted log records to a single file or to a set of rotating log files. By default, log files are written to `logs/openidm*.log` files.
- `ConsoleHandler` writes formatted logs to `System.err`.

Additional log message handlers are listed in the `logging.properties` file.

Log message format

IDM supports the two default log formatters included with Java. These are set in the `conf/logging.properties` file:

- `java.util.logging.SimpleFormatter.format` outputs a text log file that is human-readable. This is the default formatter.
- `java.util.logging.XMLFormatter` outputs logs as XML, for use in logging software that can read XML logs.

IDM extends the Java `SimpleFormatter` with the following formatting options:

`org.forgerock.openidm.logger.SanitizedThreadIdLogFormatter`

This is the default formatter for console and file logging. It extends the `SimpleFormatter` to include the thread ID of the thread that generated each message. The thread ID helps with debugging when reviewing the logs.

In the following example log excerpt, the thread ID is `[19]`:

```
[19] May 23, 2018 10:30:26.959 AM org.forgerock.openidm.repo.opendj.impl.Activator start
INFO: Registered bootstrap repository service
[19] May 23, 2018 10:30:26.960 AM org.forgerock.openidm.repo.opendj.impl.Activator start
INFO: DS bundle started
```

> **Tip**
>
> The `SanitizedThreadIdLogFormatter` also encodes all control characters (such as newline characters) using URL-encoding, to protect against log forgery. Control characters in stack traces are not encoded.

`org.forgerock.openidm.logger.ThreadIdLogFormatter`

Similar to the `SanitizedThreadIdLogFormatter`, but does not encode control characters. If you do not want to encode control characters in file and console log messages, edit the file and console handlers in `conf/logging.properties` as follows:

```
java.util.logging.FileHandler.formatter = org.forgerock.openidm.logger.ThreadIdLogFormatter

java.util.logging.ConsoleHandler.formatter = org.forgerock.openidm.logger.ThreadIdLogFormatter
```

The `SimpleFormatter` (and, by extension, the `SanitizedThreadIdLogFormatter` and `ThreadIdLogFormatter`) lets you customize what information to include in log messages, and how this information is laid out. By default, log messages include the date, time (down to the millisecond), log level, source of the message, and the message sent (including exceptions). To change the defaults, adjust the value of `java.util.logging.SimpleFormatter.format` in your `conf/logging.properties` file. For more information on how to customize the log message format, refer to the related Java documentation.

Logging level

By default, IDM logs messages at the `INFO` level. This logging level is specified with the following global property in `conf/logging.properties`:

```
.level=INFO
```

You can specify different separate logging levels for individual server features which override the global logging level. Set the log level, per package to one of the following:

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

For example, the following setting decreases the messages logged by the embedded PostgreSQL database:

```
# reduce the logging of embedded postgres since it is very verbose
ru.yandex.qatools.embed.postgresql.level = SEVERE
```

Set the log level to `OFF` to disable logging completely (Disable Logs), or to `ALL` to capture all possible log messages.

If you use `logger` functions in your JavaScript scripts, set the log level for the scripts as follows:

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

You can override the log level settings, per script, with the following setting:

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

For more information about using `logger` functions in scripts, refer to Log Functions.

Important

It is strongly recommended that you do *not* log messages at the `FINE` or `FINEST` levels in a production environment. Although these levels are useful for debugging issues in a test environment, they can result in accidental exposure of sensitive data. For example, a password change patch request can expose the updated password in the Jetty logs.

Log file rotation

By default, IDM rotates log files when the size reaches 5 MB, and retains up to 5 files. All system and custom log messages are also written to these files. You can modify these limits in the following properties in the `logging.properties` file for your project:

```
# Limiting size of output file in bytes:
java.util.logging.FileHandler.limit = 5242880

# Number of output files to cycle through, by appending an
# integer to the base file name:
java.util.logging.FileHandler.count = 5
```

Note

There is currently no `logging.properties` setting for time-based rotation of server log files. However, on UNIX systems you can use the `logrotate` command to schedule server log rotation at a regular interval. For more information, refer to the logrotate man page.

Disable logs

If necessary, you can disable logs. For example, to disable `ConsoleHandler` logging, make the following changes in your project's `conf/logging.properties` file before you start IDM.

Set `java.util.logging.ConsoleHandler.level = OFF`, and 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=...
```

Monitoring

IDM includes the following tools for monitoring metrics:

- A Dropwizard dashboard widget, for viewing metrics within IDM.

• A Prometheus endpoint, for viewing metrics through external resources such as Prometheus and Grafana.

Enable metrics

IDM does not collect metrics by default. To enable metrics collection, open `conf/metrics.json` and set the `enabled` property to `true`:

```
{
    "enabled" : true
}
```

After you have enabled metrics, the following command returns all collected metrics:

Request

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

Response


```
{
  "result": [
    {
      "_id": "repo.ds.get-connection",
      "count": 87,
      "max": 18.273865999999998,
      "mean": 2.0228276744117877,
      "min": 0.64178,
      "p50": 1.487925,
      "p75": 2.248557,
      "p95": 4.115021,
      "p98": 5.620608,
      "p99": 8.533398,
      "p999": 18.273865999999998,
      "stddev": 1.9332103308881228,
      "m15_rate": 8.653295680258273,
      "m1_rate": 5.1162605576082285,
      "m5_rate": 8.004328242732594,
      "mean_rate": 1.8443204365017944,
      "duration_units": "milliseconds",
      "rate_units": "calls/second",
      "total": 178.917206,
      "_type": "timer"
    },
    {
      "_id": "jvm.memory-usage.pools.G1-Old-Gen.committed",
      "value": 794820608,
      "_type": "gauge"
    },
    {
      "_id": "jvm.max-memory",
      "value": 2147483648,
      "_type": "gauge"
    },
    {
      "_id": "jvm.memory-usage.pools.CodeHeap-'non-profiled-nmethods'.init",
      "value": 2555904.0,
      "_type": "gauge"
    },
    {
      "_id": "scheduler.persisted.add-job",
      "count": 2,
      "max": 19.246833,
      "mean": 13.2243465,
      "min": 7.20186,
      "p50": 19.246833,
      "p75": 19.246833,
      "p95": 19.246833,
      "p98": 19.246833,
      "p99": 19.246833,
      "p999": 19.246833,
      "stddev": 6.022486499999999,
      "m15_rate": 0.38261149564121166,
      "m1_rate": 0.2053668476130369,
      "m5_rate": 0.350069327617179,
      "mean_rate": 0.042679419036704344,
      "duration_units": "milliseconds",
```

```
      "rate_units": "calls/second",
      "total": 26.448693,
      "_type": "timer"
    },
    {
      "_id": "jvm.memory-usage.pools.G1-Old-Gen.init",
      "value": 2.03423744E+9,
      "_type": "gauge"
    },
    {
      "_id": "internal.user.relationship.validate-relationship-fields",
      "count": 4,
      "max": 0.00233,
      "mean": 0.00192725,
      "min": 0.00153,
      "p50": 0.002253,
      "p75": 0.00233,
      "p95": 0.00233,
      "p98": 0.00233,
      "p99": 0.00233,
      "p999": 0.00233,
      "stddev": 0.0003660105018985111,
      "m15_rate": 0.7652229912824233,
      "m1_rate": 0.4107336952260738,
      "m5_rate": 0.700138655234358,
      "mean_rate": 0.08670015551801163,
      "duration_units": "milliseconds",
      "rate_units": "calls/second",
      "total": 0.007709,
      "_type": "timer"
    },
    {
      "_id": "jvm.memory-usage.total.max",
      "value": 2147483647,
      "_type": "gauge"
    },
    {
      "_id": "jvm.memory-usage.total.committed",
      "value": 2386423808,
      "_type": "gauge"
    },
    {
      "_id": "repo.ds.create.internal_user",
      "count": 4,
      "max": 18.306962,
      "mean": 5.7264095,
      "min": 1.4013419999999999,
      "p50": 1.6358979999999999,
      "p75": 18.306962,
      "p95": 18.306962,
      "p98": 18.306962,
      "p99": 18.306962,
      "p999": 18.306962,
      "stddev": 7.2638797944374565,
      "m15_rate": 0.7652229912824233,
      "m1_rate": 0.4107336952260738,
      "m5_rate": 0.700138655234358,
      "mean_rate": 0.08670216068319382,
      "duration_units": "milliseconds",
      "rate_units": "calls/second",
      "total": 22.905638,
      "_type": "timer"
```

```
    },
    {
      "_id": "jvm.memory-usage.heap.init",
      "value": 2147483648,
      "_type": "gauge"
    },
    {
      "_id": "repo.ds.update.cluster",
      "count": 8,
      "max": 3.679064,
      "mean": 2.9518007249923834,
      "min": 2.3344139999999998,
      "p50": 2.9840329999999997,
      "p75": 3.1550309999999997,
      "p95": 3.679064,
      "p98": 3.679064,
      "p99": 3.679064,
      "p999": 3.679064,
      "stddev": 0.3739151611284433,
      "m15_rate": 0.3923533692947412,
      "m1_rate": 0.30850549439674596,
      "m5_rate": 0.3778195015144161,
      "mean_rate": 0.2133903215631551,
      "duration_units": "milliseconds",
      "rate_units": "calls/second",
      "total": 23.459754,
      "_type": "timer"
    },
    {
      "_id": "repo.ds.read.cluster",
      "count": 8,
      "max": 2.3707599999999998,
      "mean": 1.9349245566320208,
      "min": 1.192601,
      "p50": 1.9848709999999998,
      "p75": 2.204475,
      "p95": 2.3707599999999998,
      "p98": 2.3707599999999998,
      "p99": 2.3707599999999998,
      "p999": 2.3707599999999998,
      "stddev": 0.3540316157820945,
      "m15_rate": 0.2,
      "m1_rate": 0.2,
      "m5_rate": 0.2,
      "mean_rate": 0.21335481478933183,
      "duration_units": "milliseconds",
      "rate_units": "calls/second",
      "total": 15.501201,
      "_type": "timer"
    },
    {
      "_id": "jvm.memory-usage.pools.CodeHeap-'non-nmethods'.usage",
      "value": 0.3455020215633423,
      "_type": "gauge"
    },
    {
      "_id": "jvm.memory-usage.pools.Metaspace.init",
      "value": 0.0,
      "_type": "gauge"
    },
    {
      "_id": "repo.ds.create.scheduler",
```

```
      "count": 4,
      "max": 5.671175,
      "mean": 3.72545225,
      "min": 1.702539,
      "p50": 4.15715,
      "p75": 5.671175,
      "p95": 5.671175,
      "p98": 5.671175,
      "p99": 5.671175,
      "p999": 5.671175,
      "stddev": 1.4309134647621733,
      "m15_rate": 0.7652229912824233,
      "m1_rate": 0.4107336952260738,
      "m5_rate": 0.700138655234358,
      "mean_rate": 0.08536364328879296,
      "duration_units": "milliseconds",
      "rate_units": "calls/second",
      "total": 14.901809,
      "_type": "timer"
    },
    {
      "_id": "jvm.memory-usage.pools.G1-Survivor-Space.committed",
      "value": 49283072,
      "_type": "gauge"
    },
    {
      "_id": "jvm.memory-usage.heap.usage",
      "value": 0.18285735324025154,
      "_type": "gauge"
    },
    {
      "_id": "jvm.garbage-collector.G1-Old-Generation.count",
      "value": 4.0,
      "_type": "gauge"
    },
    {
      "_id": "jvm.thread-state.waiting.count",
      "value": 54.0,
      "_type": "gauge"
    },
    {
      "_id": "jvm.available-cpus",
      "value": 16.0,
      "_type": "gauge"
    },
    {
      "_id": "jvm.class-loading.loaded",
      "value": 22506.0,
      "_type": "gauge"
    },
    {
      "_id": "null-array-filter.augmentation.read",
      "count": 10,
      "max": 0.03385,
      "mean": 0.0205321,
      "min": 0.012830999999999999,
      "p50": 0.022903999999999997,
      "p75": 0.02393,
      "p95": 0.03385,
      "p98": 0.03385,
      "p99": 0.03385,
      "p999": 0.03385,
```

```
      "stddev": 0.006245525749686731,
      "m15_rate": 1.913057478206058,
      "m1_rate": 1.0268342380651845,
      "m5_rate": 1.7503466380858956,
      "mean_rate": 0.21664554977539055,
      "duration_units": "milliseconds",
      "rate_units": "calls/second",
      "total": 0.205321,
      "_type": "timer"
    },
    {
      "_id": "filter.scripted.on-request.d6fc81179beaca37094a23c2fcd00aaf54bb3ef9:router:onRequest",
      "count": 1,
      "max": 27.052286,
      "mean": 27.052286,
      "min": 27.052286,
      "p50": 27.052286,
      "p75": 27.052286,
      "p95": 27.052286,
      "p98": 27.052286,
      "p99": 27.052286,
      "p999": 27.052286,
      "stddev": 0.0,
      "m15_rate": 0.0,
      "m1_rate": 0.0,
      "m5_rate": 0.0,
      "mean_rate": 30.733696941652937,
      "duration_units": "milliseconds",
      "rate_units": "calls/second",
      "total": 27.052286,
      "_type": "timer"
    },
    {
      "_id": "repo.ds.read.internal_user",
      "count": 4,
      "max": 0.893355,
      "mean": 0.7506262499999999,
      "min": 0.667094,
      "p50": 0.733529,
      "p75": 0.893355,
      "p95": 0.893355,
      "p98": 0.893355,
      "p99": 0.893355,
      "p999": 0.893355,
      "stddev": 0.08575225314058808,
      "m15_rate": 0.7652229912824233,
      "m1_rate": 0.4107336952260738,
      "m5_rate": 0.700138655234358,
      "mean_rate": 0.08670579236948879,
      "duration_units": "milliseconds",
      "rate_units": "calls/second",
      "total": 3.002505,
      "_type": "timer"
    },
    {
      "_id": "jvm.memory-usage.pools.CodeHeap-'non-profiled-nmethods'.committed",
      "value": 7864320.0,
      "_type": "gauge"
    },
    {
      "_id": "scheduler.job-store.repo.query-list.triggers",
      "count": 18,
```

```
        "max": 16.820667999999998,
        "mean": 3.70247311020658,
        "min": 1.068907,
        "p50": 2.205507,
        "p75": 3.3469379999999997,
        "p95": 15.960484,
        "p98": 16.820667999999998,
        "p99": 16.820667999999998,
        "p999": 16.820667999999998,
        "stddev": 4.114367764764707,
        "m15_rate": 1.5402060021782797,
        "m1_rate": 0.9277075719931202,
        "m5_rate": 1.4281843370491416,
        "mean_rate": 0.38150669470151066,
        "duration_units": "milliseconds",
        "rate_units": "calls/second",
        "total": 70.044336,
        "_type": "timer"
    },
    ...
}
```


Note

Metrics are only collected after they have been triggered by activity in IDM, such as a reconciliation.

Learn more:

- Metrics reference
- Load testing

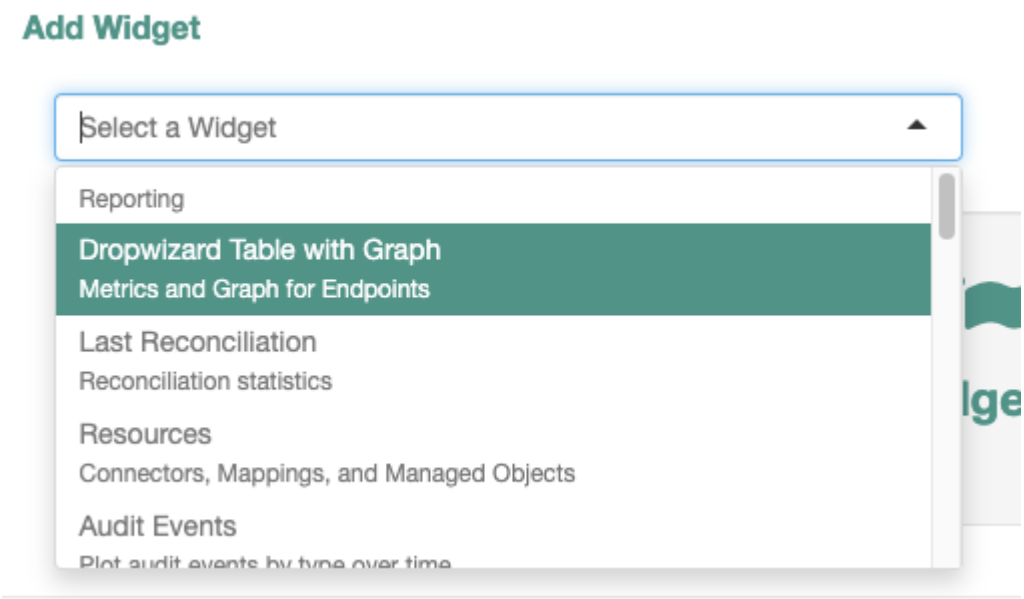
Dropwizard widget

The Dropwizard widget creates a graph of metrics based on server activity and is useful for lightweight, live monitoring of IDM. The widget has the following limitations:

- The graph created by the widget does not persist. If you reload or navigate away from the page, the graph restarts.
- The widget only works with time-based metrics.

To add the Dropwizard widget:

1. From the navigation bar, click **Dashboards > Dashboard Name**.
2. On the **Dashboard Name** page, click **Add Widget**.
3. In the **Add Widget** window, from the **Select a Widget** drop-down list, select **Dropwizard Table with Graph**.



4. To preview any metric on the graph, click **Add to Graph** adjacent to any metric.
5. Click **Add**.

   The Dropwizard widget now displays on the dashboard.

Prometheus endpoint

This topic describes how to configure Prometheus and Grafana to collect IDM metrics. These third-party tools are *not* supported by ForgeRock. Refer to the Prometheus documentation.

Prometheus is a third-party tool used for gathering and processing monitoring data. Prometheus uses the `openidm/metrics/prometheus` endpoint to gather information. This endpoint is protected by a basic authentication filter, using the following credentials, set in the `resolver/boot.properties` file:

```
openidm.prometheus.username=username
openidm.prometheus.password=password
```

Disable Prometheus

To disable IDM's Prometheus handler, comment out or remove `openidm.prometheus.username` and `openidm.prometheus.password` from the `resolver/boot.properties` file. If these properties are not set, IDM does not enable the Prometheus handler.

Configure Prometheus

1. Download Prometheus.
2. Create a `prometheus.yml` configuration file. For more information, refer to the Prometheus configuration documentation. An example `prometheus.yml` file:

```
global:
  scrape_interval: 15s
  external_labels:
    monitor: 'my_prometheus'

# https://prometheus.io/docs/operating/configuration/#scrape_config
scrape_configs:
  - job_name: 'openidm'
    scrape_interval: 15s
    scrape_timeout: 5s
    metrics_path: 'openidm/metrics/prometheus'
    scheme: http
    basic_auth:
      username: 'prometheus'
      password: 'prometheus'
    static_configs:
      - targets: ['localhost:8080']
```

This example configures Prometheus to poll the `openidm/metrics/prometheus` endpoint every 5 seconds (`scrape_interval: 5s`), receiving metrics in a plain text format (`_fields: ['text']` and `_mimeType: ['text/plain;version=0.0.4']`). For more information about reporting formats, refer to the Prometheus documentation on Exposition Formats.

3. Verify the configuration returns metric results:

Request

```
curl \
--user prometheus:prometheus \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/metrics/prometheus'
```

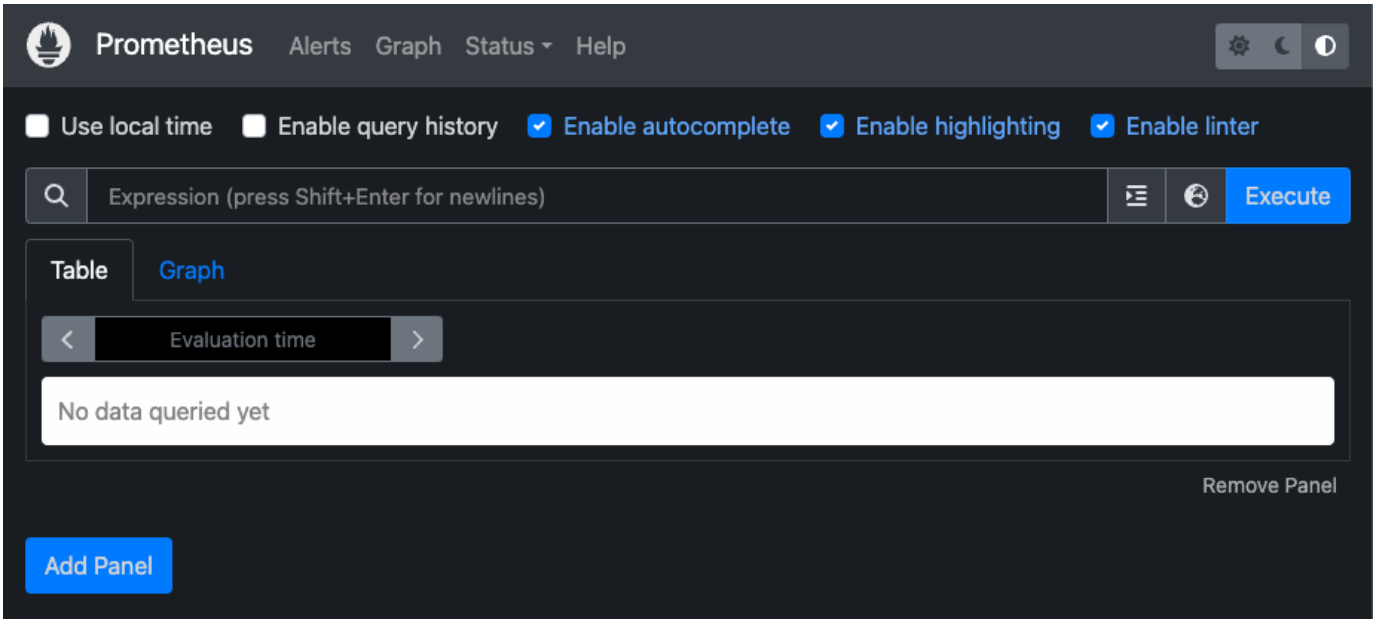
Response

```
# HELP idm_jvm_available_cpus Automatically generated
# TYPE idm_jvm_available_cpus gauge
idm_jvm_available_cpus 10.0
# HELP idm_jvm_class_loading_loaded Automatically generated
# TYPE idm_jvm_class_loading_loaded gauge
idm_jvm_class_loading_loaded 24876.0
# HELP idm_jvm_class_loading_unloaded Automatically generated
# TYPE idm_jvm_class_loading_unloaded gauge
idm_jvm_class_loading_unloaded 1.0
# HELP idm_jvm_free_used_memory_bytes Automatically generated
# TYPE idm_jvm_free_used_memory_bytes gauge
idm_jvm_free_used_memory_bytes 9.77543264E8
# HELP idm_jvm_garbage_collector_g1_old_generation_count Automatically generated
# TYPE idm_jvm_garbage_collector_g1_old_generation_count gauge
idm_jvm_garbage_collector_g1_old_generation_count 0.0
# HELP idm_jvm_garbage_collector_g1_old_generation_time Automatically generated
# TYPE idm_jvm_garbage_collector_g1_old_generation_time gauge
idm_jvm_garbage_collector_g1_old_generation_time 0.0
# HELP idm_jvm_garbage_collector_g1_young_generation_count Automatically generated
# TYPE idm_jvm_garbage_collector_g1_young_generation_count gauge
idm_jvm_garbage_collector_g1_young_generation_count 82.0
# HELP idm_jvm_garbage_collector_g1_young_generation_time Automatically generated
# TYPE idm_jvm_garbage_collector_g1_young_generation_time gauge
idm_jvm_garbage_collector_g1_young_generation_time 2127.0
# HELP idm_jvm_max_memory_bytes Automatically generated
# TYPE idm_jvm_max_memory_bytes gauge
idm_jvm_max_memory_bytes 2.147483648E9
...
```

4. Start Prometheus with the `prometheus.yml` configuration file:

```
prometheus --config.file=/path/to/prometheus.yml
```

5. To confirm that Prometheus is gathering data from IDM, go to the Prometheus monitoring page (default `http://localhost:9090`).



Configure Grafana

Prometheus lets you monitor and process information provided by IDM. If you need deeper analytics, you can use tools such as Grafana to create customized charts and graphs based on Prometheus data. For information on installing and running Grafana, refer to the Grafana website.

You can also monitor aspects of IDM's performance using Prometheus to plug JVM metrics into a Grafana dashboard. For more information on using metrics to observe the system under load, refer to Load testing.

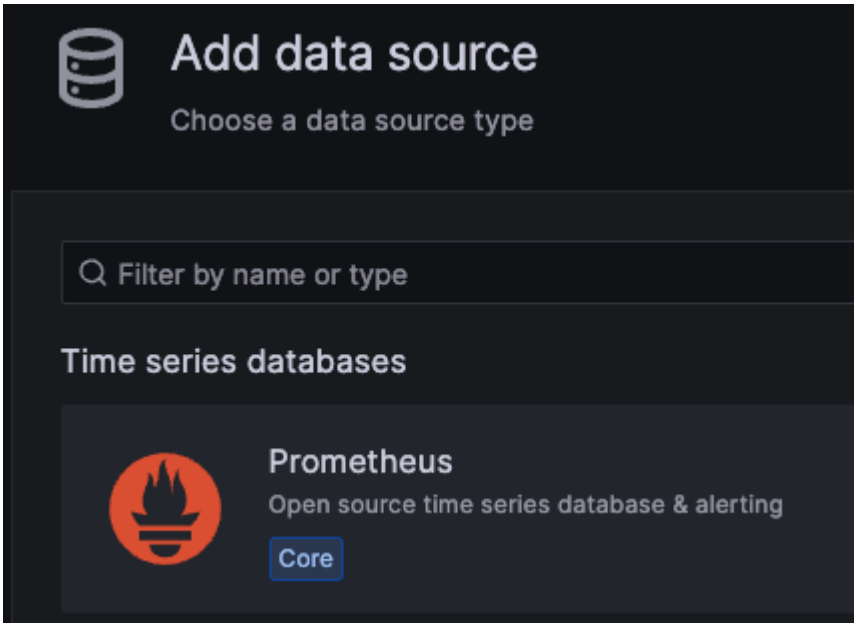
> **Tip**
>
> Before you get started, download the *Monitoring Dashboard Samples* from the Backstage download site. Open `monitoring.dashboard.json` from the downloaded .zip file, as you'll need it during the following procedure.

To set up a Grafana dashboard with IDM metrics using Prometheus:

1. In a browser, go to the main Grafana page (default `http://localhost:3000`) and log in.

   > **Tip**
   >
   > The default username and password for Grafana is `admin`.

2. To add your Prometheus installation to Grafana as a data source, click the toggle menu button, and click **Connections > Data sources**.
3. On the **Data sources** page, click **Add data source**.
4. On the **Add data source** page, select **Prometheus**.



5. Enter information and select options, as needed. The information you enter here should match the settings in the `monitoring.dashboard.json` file:
    1. Give your data source a name; for example, `ForgeRockIDM`.
    2. Set the **URL** (default `http://localhost:9090`).
    3. Enable **Basic auth**.
        1. Enter the **User** (default `prometheus`).
        2. Enter the **Password** (default `prometheus`).
6. Click **Save & test**.

    If the test succeeds, Grafana displays **Data source is working**.

Create a Grafana dashboard

After Prometheus has been configured as a data source in Grafana, you can create a dashboard with IDM metrics:

1. In Grafana, click the toggle menu button ☰, and click **Dashboards**.
2. Click **New**, and do one of the following:
    - Select **Import**.
        1. On the **Import dashboard** page, drag the `monitoring.dashboard.json` file from its location on your system to the **Upload dashboard JSON file** area.
        2. Enter information in the **Options** area, and select the Prometheus data source you previously created.
        3. Click **Import**.

- Select **New dashboard**.
    1. Click **Add visualization**.
    2. Select the Prometheus data source you previously created.
    3. Configure the panel.

> **Tip**
>
> For more information, refer to:
> - Prometheus query language
> - Panel editor overview

Load testing

Load testing can help you get the most out of IDM and other ForgeRock products. The benefits load testing provides include:

- Reducing the chance that unexpected spikes in system activity will cause the system to become unstable
- Allowing developers and system administrators to reason more accurately and be more confident in release cycle timelines
- Providing baseline statistics which can be used to identify and investigate unexpected behavior

Load testing is a complex subject that requires knowledge of your system and a disciplined approach. There is no "one-size-fits-all" solution that applies in all circumstances. However, there are some basic principles to keep in mind while planning, executing, and evaluating load tests.

Planning tests

The first step is to determine what metrics need to be examined, what components are going to be tested, what levels of load are going to be used, and what response ranges are acceptable. Answering these questions requires:

- Service-level Agreements (SLAs)
- Understanding of your use case
- Baseline knowledge of your system

SLAs provide a stationary, business-based target to aim for in testing. An example SLA appears as follows:

Service/Endpoint	Sustained load	Peak load	Required response time
Customer auth against LDAP repo	50,000 over 16 hours	4,000 per second three times in a 16-hour period	200ms
Employee auth against AD repo	4,000 over 10 hours	100/second	400ms
Customer registration	1,000 over 24 hours	10/second	500ms

Service/Endpoint	Sustained load	Peak load	Required response time
Employee password reset	10 over 24 hours	1/second	500ms

Sample SLA warnings and details:

- Response times are between load generator and ForgeRock platform and do not account for latency between client devices and architecture.
- IDM must support four writes and 45 read transactions per second for 12 hours using DS as the repository.
- IDM must support 2,000 changes from HR service.
- Measuring response times occurs after establishing 10,000 active, concurrent stateful sessions with 10,000 unique identities.

Details will vary depending on your use case and application flow, present usage patterns, full load profile, and environment. To get the most benefit, collect this information.

The system's full load profile depends on how it is designed and used. For example, some systems have thousands of clients each using a small slice of bandwidth, while others have only a few high-bandwidth connections. Understanding these nuances helps determine an appropriate number of connections and threads of execution to use to generate a test load.

Tip

If you have trouble determining which systems and components are being used at various points during your application flow, consider modeling your application using a sequence diagram.

Understanding resource use

Understanding what resources are heavily consumed by ForgeRock products will help you with your test planning. The following chart details some products and their consumed resources:

Product	Consumed resources
AM with external stores	CPU, memory
DS as a user repository	I/O, memory
DS as a token store	I/O, memory (if high token count)
IDM	I/O; CPU and memory play an important role in provisioning, sync, and user self-service
IG	CPU

All of the above depends on network performance, including name resolution and proper load balancing when required.

Executing tests

When it comes to executing tests, these are the basic principles to keep in mind:

1. Every system is different; "it depends" is the cardinal rule.
2. Testing scenarios that don't happen in reality gives you test results that don't happen in reality.
3. System performance is constrained by the scarcest resource.

One way to ensure that your tests reflect real use patterns is to begin with a load generator that creates periods of consistent use and periods of random spikes in activity. During the consistent periods, gradually add load until you exceed your SLAs and baselines. By using that data and the data from the periods of spiking activity, you can determine how your system handles spikes in activity in many different scenarios.

> **Note**
>
> Your load generator should be located on separate hardware/instances from your production systems. It should have adequate resources to generate the expected load.

When testing systems with many components, begin by testing the most basic things — I/O, CPU, and memory use. IDM provides insight into these by exposing JVM Metrics.

Once you have an understanding of the basic elements of your system, introduce new components into the tests. Keep a record of each test's environment and the components which were under test. These components may include:

- Hardware/Hypervisor/Container platform
- Hosting OS/VM/Container environment
- Hosted OS
- Java Virtual Machine (JVM)
- Web/J2EE Container (if used to host ForgeRock AM/IG or ForgeRock AM Agent)
- Databases, repositories, and directory servers used with ForgeRock
- Networking, load balancers, and firewalls between instances
- SSL, termination points, and other communications
- Points of integration, if any
- Other applications and services that utilize ForgeRock components
- Load generation configuration
- Sample data, logs from test runs, and other generated files

> **Tip**
>
> While there are many tools that can help you monitor your system, a thorough understanding of your system logs is the best path to understanding its behavior.

> **Warning**
>
> To keep your results clear and focused, only add or adjust one variable at a time.
> Do not run tests designed to stress the system to its theoretical limit. The results you get from these stress tests rarely provide actionable insights.

Metrics reference

IDM exposes a number of metrics. All metrics are available at both the `openidm/metrics/api` and `openidm/metrics/prometheus` endpoints. The actual metric names can vary, depending on the endpoint used. Also refer to Monitoring.

Metric types

Metrics are organized into the following *types*:

Timer

Timers provide a histogram of the duration of an event, along with a measure of the rate of occurrences. Timers can be monitored using the Dropwizard dashboard widget and the IDM Prometheus endpoint. Durations in timers are measured in milliseconds. Rates are reported in number of calls per second. The following example shows a *Timer* metric:


```
{
   "_id": "sync.source.perform-action",
   "count": 2,
   "max": 371.53391,
   "mean": 370.1752705,
   "min": 368.816631,
   "p50": 371.53391,
   "p75": 371.53391,
   "p95": 371.53391,
   "p98": 371.53391,
   "p99": 371.53391,
   "p999": 371.53391,
   "stddev": 1.3586395,
   "m15_rate": 0.393388581528647,
   "m1_rate": 0.311520313228562,
   "m5_rate": 0.3804917698002856,
   "mean_rate": 0.08572717156016606,
   "duration_units": "milliseconds",
   "rate_units": "calls/second",
   "total": 740.350541,
   "_type": "timer"
 }
```


Summary

Summaries are similar to Timers in that they measure a distribution of events. However, Summaries record values that aren't units of time, such as user login counts. Summaries cannot be graphed in the Dropwizard dashboard widget, but are available through the Prometheus endpoint, and by querying the `openidm/metrics/api` endpoint directly. The following example shows a *Summary* metric:

```
{
  "_id": "audit.recon",
  "m15_rate": 0.786777163057294,
  "m1_rate": 0.623040626457124,
  "m5_rate": 0.7609835396005712,
  "mean_rate": 0.16977218861919927,
  "units": "events/second",
  "total": 4,
  "count": 4,
  "_type": "summary"
}
```

Gauge

Gauge metrics return a numerical value that can increase or decrease. The value for a gauge is calculated on request, and represents the state of the metric at that specific time. The following example shows a *Gauge* metric:

```
{
  "_id": "jvm.used-memory",
  "value": 2147483648,
  "_type": "gauge"
}
```

API metrics

Metrics accessed at the `api` endpoint (such as those consumed by the Dropwizard dashboard widget) use dot notation for their metric names; for example, `recon.target-phase`. The following table lists the API metrics available in IDM:

API metrics available in IDM

API Metric Name	Type	Description
`audit.audit-topic`	Summary	Count of all audit events generated of a given topic type.
`field.augmentation.edge`	Timer	Rate of reading response objects, to fulfill the `_fields` requested (when the fields were not populated by the initial repo query).
`field.augmentation.vertex`	Timer	Rate of reading response objects, to fulfill the `_fields` requested (when the fields were not populated by the initial repo query).
`filter.filter-type.action.script-name`	Timer	Rate at which filter scripts are executed, per action. Monitors scripted filters and delegated admin.

API Metric Name	Type	Description
`icf.system-identifier.objectClass.query._queryExpression`	Timer	Rate of ICF query executions with queryExpression, and time taken to perform this operation.
`icf.system-identifier.objectClass.query._queryFilter`	Timer	Rate of ICF query executions with queryFilter, and time taken to perform this operation.
`icf.system-identifier.objectClass.query._queryId.queryId`	Timer	Rate of ICF query executions with queryId, and time taken to perform this operation.
`icf.system-identifier.objectClass.query._UNKNOWN`	Timer	Rate of ICF query executions when the query type is UNKNOWN, and time taken to perform this operation.
`internal.managed-object.operation`	Timer	Rate of operations on internal objects.
`internal.managed-object.relationship.fetch-relationship-fields`	Timer	Rate of fetch operations of relationship fields for internal objects.
`internal.managed-object.relationship.get-relationship-value-for-resource`	Timer	Query rate on relationship values for internal objects.
`internal.managed-object.script.script-name`	Timer	Rate of script executions on internal object.
`internal.managed-object.relationship.validate-relationship-fields`	Timer	Rate of validate operations of relationship fields for internal objects.
`managed.field.augmentation`	Timer	Rate of responses requiring field augmentation. When the repository cannot retrieve all data in a single call, IDM performs additional read operations to complete (augment) the missing data.
`managed.managed-object.operation`	Timer	Rate of operations on a managed object.
`managed.managed-object.relationship.fetch-relationship-fields`	Timer	Rate of fetches of relationship fields of a managed object.
`managed.managed-object.relationship.get-relationship-value-for-resource`	Timer	Rate of queries to get relationship values for a resource on a managed object.
`managed.managed-object.relationship.validate-relationship-fields`	Timer	Rate of validations of relationship fields of a managed object.
`managed.managed-object.script.script-name`	Timer	Rate of executions of a script on a managed object.

API Metric Name	Type	Description
`managed.object.handle-temporal-constraints-on-create`	Timer	Latency of enforcing temporal constraints on role objects during object creation.
`managed.object.handle-temporal-constraints-on-delete`	Timer	Latency of enforcing temporal constraints on role objects during object deletion.
`managed.object.handle-temporal-constraints-on-update`	Timer	Latency of enforcing temporal constraints on role objects during object update.
`managed.relationship.handle-temporal-constraints-on-create`	Timer	Latency of enforcing temporal constraints on relationship grants during edge creation.
`managed.relationship.handle-temporal-constraints-on-delete`	Timer	Latency of enforcing temporal constraints on relationship grants during edge deletion.
`managed.relationship.handle-temporal-constraints-on-update`	Timer	Latency of enforcing temporal constraints on relationship grants during edge update.
`managed.relationship.validate.read-relationship-endpoint-edges`	Timer	Rate of reads on relationship endpoint edges for validation.
`null_array_filter.augmentationrequestType`	Timer	Time spent in filter which maps non-nullable, null-valued array fields to an empty array. This filter is traversed for all repo access relating to internal and managed objects.
`recon`	Timer	Rate of executions of a full reconciliation, and time taken to perform this operation.
`recon-assoc-entry.merged-query.merge-results`	Timer	Rate of merge operations after source and/or target objects have been retrieved during a merged query of recon association entries.
`recon-assoc-entry.merged-query.page-assoc-entries`	Timer	Rate of individual paged recon association entry queries during a merged query. More than one page of entries might be requested to build a single page of merged results.
`recon-assoc-entry.merged-query.query-source`	Timer	Rate of source object retrieval via query when merging source objects to recon association entries.

API Metric Name	Type	Description
`recon-assoc-entry.merged-query.query-target`	Timer	Rate of target object retrieval via query when merging target objects to recon association entries.
`recon.association-persistence.recon-id-operation`	Timer	The time taken to persist association data. The operation can be `source`, `target`, or `amendsource`, depending on whether data is being produced for a source-phase or target-phase recon association, or to amend the association for a specific source.
`recon.id-queries-phase`	Timer	Rate of executions of the id query phase of a reconciliation, and time taken to perform this operation.
`recon.source-phase`	Timer	Rate of executions of the source phase of a reconciliation, and time taken to perform this operation.
`recon.source-phase.page`	Timer	Rate of pagination executions of the source phase of a reconciliation, and time taken to perform this operation.
`recon.target-phase`	Timer	Rate of executions of the target phase of a reconciliation, and time taken to perform this operation.
`repo.jdbc.relationship.edge.execute.joinedToVertex`	Timer	Time (ms) spent running the Edge→Vertex relationship join query on the database and collecting the result set.
`repo.jdbc.relationship.execute`	Timer	Rate of relationship graph query execution times.
`repo.jdbc.relationship.process`	Timer	Rate of relationship graph query result processing times.
`repo.raw._queryId.queryId`	Timer	Rate of executions of a query with queryId at a repository level, and time taken to perform this operation.

API Metric Name	Type	Description
`repo.repo-type.cache.objecttypes.event.resource-mapping`	Count	Counts the usage statistics of the `objecttypeid` cache, which maps an object type to its `objecttypeid`. The expected count is a small number of misses (sometimes, only one) and the remainder of hits.
`repo.repo-typeget-connection`	Timer	Rate of retrievals of a repository connection.
`repo.repo-type.operation.action_name.command.resource-mapping`	Timer	Rate of actions to a repository datasource for a generic/explicit mapped table.
`repo.repo-type.operation._adhoc-expression.relationship`	Timer	Rate of filtered queries (using native query expressions) on the relationship table. This metric measures the time spent making the query (in ms), and the number of times the query is invoked.
`repo.repo-type.operation._adhoc-filter.relationship`	Timer	Rate of filtered queries (using the `_queryFilter` parameter) on the relationship table. This metric measures the time spent making the query (in ms), and the number of times the query is invoked.
`repo.repo-type.create_properties.execute.resource-mapping`	Timer	Rate of execution time on the JDBC database for the `create_properties` operations. This operation is performed for every generic object `create` when it persists the searchable properties. The rate measured here does not include the time taken to obtain a connection to the database from the connection pool. The physical connections to the database have already been established inside the connection pool.
`repo.repo-type.operation.execute.resource-mapping`	Timer	Rate of execution time on the JDBC database for CRUD operations. This rate does not include the time taken to obtain a connection to the database from the connection pool. The physical connections to the database have already been established inside the connection pool.

API Metric Name	Type	Description
`repo.repo-type.query.execute.resource-mappingqueryType.]`	Timer	Rate of execution time on the JDBC database for queries (either `queryFilter` or `queryId`). This rate does not include the time taken to obtain a connection to the database from the connection pool. The physical connections to the database have already been established inside the connection pool.
`repo.repo-type.operation.relationship`	Timer	Rate of CRUDPAQ operations to a repository datasource for a generic/explicit/relationship mapped table.
`repo.repo-type.operation.relationship.stage.origin_type`	Timer	Time (ms) spent in the various phases to retrieve relationship expanded data referenced by queried objects.
`repo.repo-type.operation.resource-mapping`	Timer	Rate of initiations of a CRUDPAQ operation to a repository datasource.
`router.path-name.action.action-type`	Timer	Rate of actions over the router, and time taken to perform this operation.
`router.path-name.create`	Timer	Rate of creates over the router, and time taken to perform this operation.
`router.path-name.delete`	Timer	Rate of deletes over the router, and time taken to perform this operation.
`router.path-name.patch`	Timer	Rate of patches over the router, and time taken to perform this operation.
`router.path-name.query.queryExpression`	Timer	Rate of queries with queryExpression completed over the router, and time taken to perform this operation.
`router.path-name.query.queryFilter`	Timer	Rate of queries with queryFilter completed over the router, and time taken to perform this operation.
`router.path-name.read`	Timer	Rate of reads over the router, and time taken to perform this operation.
`router.path-name.update`	Timer	Rate of updates over the router, and time taken to perform this operation.
`scheduler.job-store.repo.operation.scheduler-object`	Timer	Time spent storing scheduled jobs in the repository.

API Metric Name	Type	Description
`scheduler.type.operation`	Timer	Execution rate of scheduler requests.
`script.script-name.request-type`	Timer	Rate of calls to a script and time taken to complete.
`selfservice.user.password.reset`	Summary	Count of all successful user self-service password resets.
`selfservice.user.registration.registration-type`	Summary	Count of all successful user self-service registrations by registration type.
`selfservice.user.registration.registration-type.provider`	Summary	Count of all successful user self-service registrations by registration type and provider.
`sync.create-object`	Timer	Rate of requests to create a target object, and time taken to perform the operation.
`sync.delete-target`	Timer	Rate of requests to delete a target object, and time taken to perform the operation.
`sync.objectmapping.mapping-name`	Timer	Rate of configurations applied to a mapping.
`sync.queue.mapping-name.action.acquire`	Timer	Rate of acquisition of queued synchronization events from the queue.
`sync.queue.mapping-name.action.discard`	Timer	Rate of deletion of synchronization events from the queue.
`sync.queue.mapping-name.action.execution`	Timer	Rate at which queued synchronization operations are executed.
`sync.queue.mapping-name.action.failed`	Summary	Number of queued synchronization operations that failed.
`sync.queue.mapping-name.action.precondition-failed`	Summary	Number of queued synchronization events that were acquired by another node in the cluster.
`sync.queue.mapping-name.action.rejected-executions`	Summary	Number of queued synchronization events that were rejected because the backing thread-pool queue was at full capacity and the thread-pool had already allocated its maximum configured number of threads.
`sync.queue.mapping-name.action.release`	Timer	Rate at which queued synchronization events are released.

API Metric Name	Type	Description
`sync.queue.mapping-name.action.release-for-retry`	Timer	Times the release of queued synchronization events after a failure and before exceeding the retry count.
`sync.queue.mapping-name.action.submit`	Timer	Rate of insertion of synchronization events into the queue.
`sync.queue.mapping-name.poll-pending-events`	Timer	The latency involved in polling for synchronization events.
`sync.raw-read-object`	Timer	Rate of reads of an object.
`sync.source.assess-situation`	Timer	Rate of assessments of a synchronization situation.
`sync.source.correlate-target`	Timer	Rate of correlations between a target and a given source, and time taken to perform this operation.
`sync.source.determine-action`	Timer	Rate of determinations done on a synchronization action based on its current situation.
`sync.source.perform-action`	Timer	Rate of completions of an action performed on a synchronization operation.
`sync.target.assess-situation`	Timer	Rate of assessments of a target situation.
`sync.target.determine-action`	Timer	Rate of determinations done on a target action based on its current situation.
`sync.target.perform-action`	Timer	Rate of completions of an action performed on a target sync operation.
`sync.update-target`	Timer	Rate of requests to update an object on the target, and the time taken to perform this operation.
`user.login.user-type`	Summary	Count of all successful logins by user type.
`user.login.user-type.provider`	Summary	Count of all successful logins by user type and provider.

API Metric Name	Type	Description
`virtual-properties-from-relationships.not-found.virtual_properties.resource_collection_relationship_field`	Summary	Number of 404 responses encountered when querying the `resource_collection` / `relationship_field` specified in the traversal_depthX tag for the most recent X.
`virtual-properties-from-relationships.unsatisified-temp-constraint.virtual_properties.resource_collection_relationship_field`	Summary	Number of edges skipped due to an unsatisfied temporal constraint on either the edge or the referred-to vertex. Encountered when querying the resource collection and relationship field at the traversal_depthX tag for the most recent X.
`virtual-properties-from-relationships.virtual_properties.resource_collection_relationship_field`	Timer	Time spent traversing relationship fields to calculate the specified virtual properties. The managed objects linked to by the traversal relationship fields define a tree, whose root is the virtual property host. This object tree is traversed depth-first, with the traversal_depthX corresponding to the latency involved with each relationship traversal. Traversal_depth0 corresponds to the first relationship field traversed. Because the tree is traversed depth-first, traversal_depthX will subsume all the traversal latencies for all traversal_depth Y, where Y>X.

API JVM metrics available in IDM

> **Note**
>
> These metrics depend on the JVM version and configuration. In particular, garbage-collector-related metrics depend on the garbage collector that the server uses. The garbage-collector metric names are unstable, and can change even in a minor JVM release.

API Metric Name	Type	Unit	Description
`jvm.available-cpus`	Gauge	Count	Number of processors available to the JVM. For more information, refer to Runtime.

API Metric Name	Type	Unit	Description
`jvm.class-loading.loaded`	Gauge	Count	Number of classes loaded since the Java virtual machine started. For more information, refer to ClassLoadingMXBean.
`jvm.class-loading.unloaded`	Gauge	Count	Number of classes unloaded since the Java virtual machine started. For more information, refer to ClassLoadingMXBean.
`jvm.free-used-memory`	Gauge	Bytes	For more information, refer to Runtime.
`jvm.garbage-collector.G1-Old-Generation.count`	Gauge	Count	For each garbage collector in the JVM. For more information, refer to GarbageCollectorMXBean.
`jvm.garbage-collector.G1-Old-Generation.time`	Gauge	Milliseconds	
`jvm.garbage-collector.G1-Young-Generation.count`	Gauge	Count	
`jvm.garbage-collector.G1-Young-Generation.time`	Gauge	Milliseconds	
`jvm.max-memory`	Gauge	Bytes	For more information, refer to Runtime.
`jvm.memory-usage.heap.committed`	Gauge	Bytes	Amount of heap memory committed for the JVM to use. For more information, refer to MemoryMXBean.
`jvm.memory-usage.heap.init`	Gauge	Bytes	
`jvm.memory-usage.heap.max`	Gauge	Bytes	Maximum amount of heap memory available to the JVM.
`jvm.memory-usage.heap.usage`	Gauge	Bytes	
`jvm.memory-usage.heap.used`	Gauge	Bytes	Amount of heap memory used by the JVM.
`jvm.memory-usage.non-heap.committed`	Gauge	Bytes	Amount of non-heap memory committed for the JVM to use.
`jvm.memory-usage.non-heap.init`	Gauge	Bytes	Amount of non-heap memory the JVM initially requested from the operating system.

API Metric Name	Type	Unit	Description
`jvm.memory-usage.non-heap.max`	Gauge	Bytes	Maximum amount of non-heap memory available to the JVM.
`jvm.memory-usage.non-heap.usage`	Gauge	Bytes	
`jvm.memory-usage.non-heap.used`	Gauge	Bytes	Amount of non-heap memory used by the JVM.
`jvm.memory-usage.pools.CodeHeap-'non-nmethods'.committed`	Gauge	Bytes	For each pool. For more information, refer to MemoryPoolMXBean.
`jvm.memory-usage.pools.CodeHeap-'non-nmethods'.init`	Gauge	Bytes	
`jvm.memory-usage.pools.CodeHeap-'non-nmethods'.max`	Gauge	Bytes	
`jvm.memory-usage.pools.CodeHeap-'non-nmethods'.usage`	Gauge	Bytes	
`jvm.memory-usage.pools.CodeHeap-'non-nmethods'.used`	Gauge	Bytes	
`jvm.memory-usage.pools.CodeHeap-'non-profiled-nmethods'.committed`	Gauge	Bytes	
`jvm.memory-usage.pools.CodeHeap-'non-profiled-nmethods'.init`	Gauge	Bytes	
`jvm.memory-usage.pools.CodeHeap-'non-profiled-nmethods'.max`	Gauge	Bytes	
`jvm.memory-usage.pools.CodeHeap-'non-profiled-nmethods'.usage`	Gauge	Bytes	
`jvm.memory-usage.pools.CodeHeap-'non-profiled-nmethods'.used`	Gauge	Bytes	
`jvm.memory-usage.pools.CodeHeap-'profiled-nmethods'.committed`	Gauge	Bytes	
`jvm.memory-usage.pools.CodeHeap-'profiled-nmethods'.init`	Gauge	Bytes	
`jvm.memory-usage.pools.CodeHeap-'profiled-nmethods'.max`	Gauge	Bytes	
`jvm.memory-usage.pools.CodeHeap-'profiled-nmethods'.usage`	Gauge	Bytes	

API Metric Name	Type	Unit	Description
`jvm.memory-usage.pools.CodeHeap-'profiled-nmethods'.used`	Gauge	Bytes	
`jvm.memory-usage.pools.Compressed-Class-Space.committed`	Gauge	Bytes	
`jvm.memory-usage.pools.Compressed-Class-Space.init`	Gauge	Bytes	
`jvm.memory-usage.pools.Compressed-Class-Space.max`	Gauge	Bytes	
`jvm.memory-usage.pools.Compressed-Class-Space.usage`	Gauge	Bytes	
`jvm.memory-usage.pools.Compressed-Class-Space.used`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Eden-Space.committed`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Eden-Space.init`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Eden-Space.max`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Eden-Space.usage`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Eden-Space.used`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Eden-Space.used-after-gc`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Old-Gen.committed`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Old-Gen.init`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Old-Gen.max`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Old-Gen.usage`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Old-Gen.used`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Old-Gen.used-after-gc`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Survivor-Space.committed`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Survivor-Space.init`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Survivor-Space.max`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Survivor-Space.usage`	Gauge	Bytes	
`jvm.memory-usage.pools.G1-Survivor-Space.used`	Gauge	Bytes	

API Metric Name	Type	Unit	Description
`jvm.memory-usage.pools.G1-Survivor-Space.used-after-gc`	Gauge	Bytes	
`jvm.memory-usage.pools.Metaspace.committed`	Gauge	Bytes	
`jvm.memory-usage.pools.Metaspace.init`	Gauge	Bytes	
`jvm.memory-usage.pools.Metaspace.max`	Gauge	Bytes	
`jvm.memory-usage.pools.Metaspace.usage`	Gauge	Bytes	
`jvm.memory-usage.pools.Metaspace.used`	Gauge	Bytes	
`jvm.memory-usage.total.committed`	Gauge	Bytes	Amount of memory that is committed for the JVM to use. For more information, refer to MemoryMXBean.
`jvm.memory-usage.total.init`	Gauge	Bytes	
`jvm.memory-usage.total.max`	Gauge	Bytes	
`jvm.memory-usage.total.used`	Gauge	Bytes	
`jvm.thread-state.blocked.count`	Gauge	Count	For more information, refer to ThreadMXBean.
`jvm.thread-state.count`	Gauge	Count	Number of live threads including both daemon and non-daemon threads.
`jvm.thread-state.daemon.count`	Gauge	Count	Number of live daemon threads.
`jvm.thread-state.new.count`	Gauge	Count	Number of threads in the `NEW` state.
`jvm.thread-state.runnable.count`	Gauge	Count	Number of threads in the `RUNNABLE` state.
`jvm.thread-state.terminated.count`	Gauge	Count	Number of threads in the `TERMINATED` state.
`jvm.thread-state.timed_waiting.count`	Gauge	Count	Number of threads in the `TIMED_WAITING` state.
`jvm.thread-state.waiting.count`	Gauge	Count	Number of threads in the `WAITING` state.

API Metric Name	Type	Unit	Description
`jvm.used-memory`	Gauge	Bytes	For more information, refer to totalMemory().

API workflow metrics available in IDM

API Metric Name	Type	Description
`workflow.execution.action.message`	Timer	Time spent invoking a message event.
`workflow.execution.action.signal`	Timer	Time spent invoking a signal event.
`workflow.execution.action.trigger`	Timer	Time spent triggering an execution.
`workflow.execution.query`	Timer	Time spent querying executions.
`workflow.job.action.execute`	Timer	Time spent forcing synchronous execution of a job.
`workflow.job.action.stacktrace`	Timer	Time spent displaying the stacktrace for a job that triggered an exception.
`workflow.job.delete`	Timer	Time spent deleting a job.
`workflow.job.query`	Timer	Time spent querying jobs.
`workflow.job.read`	Timer	Time spent reading a single job.
`workflow.jobdeadletter.action.execute`	Timer	Time spent to execute dead-letter job.
`workflow.jobdeadletter.action.stacktrace`	Timer	Time spent to retrieve the stacktrace for a dead-letter job.
`workflow.jobdeadletter.delete`	Timer	Time spent to delete a dead letter job.
`workflow.jobdeadletter.query`	Timer	Time spent to query dead letter jobs.
`workflow.jobdeadletter.read`	Timer	Time spent to read a dead letter job.
`workflow.model.action.deploy`	Timer	Time spent to deploy a model.
`workflow.model.action.list_deployments`	Timer	Time spent to list model deployments.
`workflow.model.action.validate_bpmn`	Timer	Time spent to validate BPMN content.
`workflow.model.create`	Timer	Time spent to create a model.
`workflow.model.delete`	Timer	Time spent to delete a model.

API Metric Name	Type	Description
`workflow.model.query`	Timer	Time spent to query models.
`workflow.model.read`	Timer	Time spent to read a model.
`workflow.model.update`	Timer	Time spent to update a model.
`workflow.processdefinition.delete`	Timer	Time spent to delete a process definition.
`workflow.processdefinition.query`	Timer	Time spent to query process definitions.
`workflow.processdefinition.read`	Timer	Time spent to read a process definition.
`workflow.processinstance.action.migrate`	Timer	Time spent to migrate a process instance.
`workflow.processinstance.action.validateMigration`	Timer	Time spent to validate a migration of a process instance.
`workflow.processinstance.create`	Timer	Time spent to create a process instance.
`workflow.processinstance.delete`	Timer	Time spent to delete a process instance.
`workflow.processinstance.query`	Timer	Time spent to query process instances.
`workflow.processinstance.read`	Timer	Time spent to read a process instance.
`workflow.taskdefinition.query`	Timer	Time spent to query task definitions.
`workflow.taskdefinition.read`	Timer	Time spent to read a task definition.
`workflow.taskinstance.action.complete`	Timer	Time spent to complete a task instance.
`workflow.taskinstance.query`	Timer	Time spent to query task instances.
`workflow.taskinstance.read`	Timer	Time spent to read a task instance.
`workflow.taskinstance.update`	Timer	Time spent to update a task instance.

Prometheus metrics

Metrics accessed through the Prometheus endpoint are prepended with `idm_` and use underscores between words; for example, `idm_recon_target_phase_seconds`. The following table lists the Prometheus metrics available in IDM:

Prometheus metrics available in IDM

Prometheus Metric Name	Type	Description
`idm_audit{audit_topic=audit-topic}`	Summary	Count of all audit events generated of a given topic type.
`idm_field_augmentation{origin-type=edge}`	Timer	Rate of reading response objects, to fulfill the `_fields` requested (when the fields were not populated by the initial repo query).
`idm_field_augmentation{origin-type=vertex}`	Timer	Rate of reading response objects, to fulfill the `_fields` requested (when the fields were not populated by the initial repo query).
`idm_filter_seconds{action=action,filter_type=filter-type,script_name=script-name}`	Timer	Rate at which filter scripts are executed, per action. Monitors scripted filters and delegated admin.
`idm_icf_system-identifier_objectClass_query__queryExpression_seconds`	Timer	Rate of ICF query executions with queryExpression, and time taken to perform this operation.
`idm_icf_system-identifier_objectClass_query__queryFilter_seconds`	Timer	Rate of ICF query executions with queryFilter, and time taken to perform this operation.
`idm_icf_system-identifier_objectClassquery__queryIdqueryId_seconds`	Timer	Rate of ICF query executions with queryId, and time taken to perform this operation.
`idm_icf_system-identifier_objectClass_query__UNKNOWN_seconds`	Timer	Rate of ICF query executions when the query type is UNKNOWN, and time taken to perform this operation.
`idm_internal_managed-object_relationship_fetch_relationship_fields_seconds`	Timer	Rate of fetch operations of relationship fields for internal objects.
`idm_internal_managed-object_relationship_get_relationship_value_for_resource_seconds`	Timer	Query rate on relationship values for internal objects.

Prometheus Metric Name	Type	Description
`idm_internal_managed-object_relationship_validate_relationship_fields_seconds`	Timer	Rate of validate operations of relationship fields for internal objects.
`idm_internal_managed-objectscriptscript-name_seconds`	Timer	Rate of script executions on internal objects.
`idm_internal_seconds{managed_object=managed-object,operation=operation>}`	Timer	Rate of operations on internal objects.
`idm_managed_field_augmentation_seconds`	Timer	Rate of responses requiring field augmentation. When the repository is unable to retrieve all the data in a single call, IDM performs additional read operations to complete (augment) the missing data.
`idm_managed_managed-object_relationship_fetch_relationship_fields_seconds`	Timer	Rate of fetches of relationship fields of a managed object.
`idm_managed_managed-object_relationship_get_relationship_value_for_resource_seconds`	Timer	Rate of queries to get relationship values for a resource on a managed object.
`idm_managed_managed-object_relationship_validate_relationship_fields_seconds`	Timer	Rate of validations of relationship fields of a managed object.
`idm_managed_managed-objectscriptscript-name_seconds`	Timer	Rate of executions of a script on a managed object.
`idm_managed_object_handle_temporal_constraints_on_create`	Timer	Latency of enforcing temporal constraints on role objects during object creation.
`idm_managed_object_handle_temporal_constraints_on_delete`	Timer	Latency of enforcing temporal constraints on role objects during object deletion.
`idm_managed_object_handle_temporal_constraints_on_update`	Timer	Latency of enforcing temporal constraints on role objects during object update.
`idm_managed_relationship_handle_temporal_constraints_on_create`	Timer	Latency of enforcing temporal constraints on relationship grants during edge creation.

Prometheus Metric Name	Type	Description
`idm_managed_relationship_handle_temporal_constraints_on_delete`	Timer	Latency of enforcing temporal constraints on relationship grants during edge deletion.
`idm_managed_relationship_handle_temporal_constraints_on_update`	Timer	Latency of enforcing temporal constraints on relationship grants during edge update.
`idm_managed_relationship_validate_read_relationship_endpoint_edges_seconds`	Timer	Rate of reads on relationship endpoint edges for validation.
`idm_managed_seconds{managed_object=managed-object,operation=operation}`	Timer	Rate of operations on a managed object.
`idm_null_array_filter.augmentationrequestType`	Timer	Time spent in filter which maps non-nullable, null-valued array fields to an empty array. This filter is traversed for all repo access relating to internal and managed objects.
`idm_recon-assoc-entry_merged-query_merge-results`	Timer	Rate of merge operations after source and/or target objects have been retrieved during a merged query of recon association entries.
`idm_recon-assoc-entry_merged-query_page-assoc-entries`	Timer	Rate of individual paged recon association entry queries during a merged query. More than one page of entries might be requested to build a single page of merged results.
`idm_recon-assoc-entry_merged-query_query-source`	Timer	Rate of source object retrieval via query when merging source objects to recon association entries.
`idm_recon-assoc-entry_merged-query_query-target`	Timer	Rate of target object retrieval via query when merging target objects to recon association entries.

Prometheus Metric Name	Type	Description
`idm_recon_association-persistence{recon-id=reconId,operation=operation}`	Timer	The time taken to persist association data. The operation can be `source`, `target`, or `amendsource`, depending on whether data is being produced for a source-phase or target-phase recon association, or to amend the association for a specific source.
`idm_recon_id_queries_phase_seconds`	Timer	Rate of executions of the id query phase of a reconciliation, and time taken to perform this operation.
`idm_recon_seconds`	Timer	Rate of executions of a full reconciliation, and time taken to perform this operation.
`idm_recon_source_phase_page_seconds`	Timer	Rate of pagination executions of the source phase of a reconciliation, and time taken to perform this operation.
`idm_recon_source_phase_seconds`	Timer	Rate of executions of the source phase of a reconciliation, and time taken to perform this operation.
`idm_recon_target_phase_seconds`	Timer	Rate of executions of the target phase of a reconciliation, and time taken to perform this operation.
`idm_repo_adhoc-expression_relationship_seconds{operation=operation,repo_type=repo-type}`	Timer	Rate of filtered queries (using native query expressions) on the relationship table. This metric measures the time spent making the query (in ms), and the number of times the query is invoked.

Prometheus Metric Name	Type	Description
`idm_repo_adhoc-filter_relationship_seconds{operation=operation,repo_type=repo-type}`	Timer	Rate of filtered queries (using the `_queryFilter` parameter) on the relationship table. This metric measures the time spent making the query (in ms), and the number of times the query is invoked.
`idm_repo_execute_seconds{operation=create_properties,repo_type=repo-type,resource_mapping=resource-mapping}`	Timer	Rate of execution time on the JDBC database for the `create_properties` operations. This operation is performed for every generic object `create` when it persists the searchable properties. The rate measured here does not include the time taken to obtain a connection to the database from the connection pool. The physical connections to the database have already been established inside the connection pool.
`idm_repo_execute_seconds{operation=operation,repo_type=repo-type,resource_mapping=resource-mapping}`	Timer	Rate of execution time on the JDBC database for CRUD operations. This rate does not include the time taken to obtain a connection to the database from the connection pool. The physical connections to the database have already been established inside the connection pool.
`idm_repo_execute_seconds{operation="query",queryType=queryFilter|queryId,repo_type=repo-type,resource_mapping=resource-mapping}`	Timer	Rate of execution time on the JDBC database for queries (either `queryFilter` or `queryId`). This rate does not include the time taken to obtain a connection to the database from the connection pool. The physical connections to the database have already been established inside the connection pool.

Prometheus Metric Name	Type	Description
`idm_repo_get_connection_seconds{repo_type=repo-type}`	Timer	Rate of retrievals of a repository connection.
`idm_repo_jdbc_cache_objecttypes_count{event="hit|miss",type=resource-mapping`	Count	Counts the usage statistics of the `objecttypeid` cache, which maps an object type to its `objecttypeid`. The expected count is a small number of misses (sometimes, only one) and the remainder of hits.
`idm_repo_jdbc_relationship_edge_execute_seconds{joinedToVertex=joinedToVertex>`	Timer	Time (ms) spent running the Edge→Vertex relationship join query on the database and collecting the result set.
`idm_repo_jdbc_relationship_execute_seconds`	Timer	Rate of relationship graph query execution times.
`idm_repo_jdbc_relationship_process_seconds`	Timer	Rate of relationship graph query result processing times.
`idm_repo_raw__queryid_credential_queryId_seconds`	Timer	Rate of executions of a query with queryId at a repository level, and time taken to perform this operation.
`idm_repo_relationship_count{operation=operation,origin_type=origin_type,repo_type=repo_type,stage=stage}`	Timer	Time (ms) spent in the various phases to retrieve relationship expanded data referenced by queried objects.
`idm_repo_relationship_seconds{operation=operation,repo_type=repo-type}`	Timer	Rate of CRUDPAQ operations to a repository datasource for a generic/explicit/relationship mapped table.
`idm_repo_seconds{action_name=action-name,command=command,operation=operation,repo_type=repo-type,resource_mapping=resource-mapping}`	Timer	Rate of actions to a repository datasource for a generic/explicit mapped table.
`idm_repo_seconds{operation=operation,repo_type=repo-type,resource_mapping=resource-mapping}`	Timer	Rate of initiations of a CRUDPAQ operation to a repository datasource.

Prometheus Metric Name	Type	Description
`idm_router_path-nameactionaction-type_seconds`	Timer	Rate of actions over the router, and time taken to perform this operation.
`idm_router_path-name_create_seconds`	Timer	Rate of creates over the router, and time taken to perform this operation.
`idm_router_path-name_delete_seconds`	Timer	Rate of deletes over the router, and time taken to perform this operation.
`idm_router_path-name_patch_seconds`	Timer	Rate of patches over the router, and time taken to perform this operation.
`idm_router_path-name_query_queryExpression_seconds`	Timer	Rate of queries with queryExpression completed over the router, and time taken to perform this operation.
`idm_router_path-name_query_queryFilter_seconds`	Timer	Rate of queries with queryFilter completed over the router, and time taken to perform this operation.
`idm_router_path-name_read_seconds`	Timer	Rate of reads over the router, and time taken to perform this operation.
`idm_router_path-name_update_seconds`	Timer	Rate of updates over the router, and time taken to perform this operation.
`idm_scheduler_job_store_repo_seconds{operation=operation,scheduler_objectscheduler-object}`	Timer	Time spent storing scheduled jobs in the repository.
`idm_scheduler_seconds{operation=operation,type=type}`	Timer	Execution rate of scheduler requests.
`idm_script_script-name_request-type`	Timer	Rate of calls to a script and time taken to complete.
`idm_selfservice_user_password_reset`	Summary	Count of all successful user self-service password resets.

Prometheus Metric Name	Type	Description
`idm_selfservice_user_registration{provider=provider,reg_type=registration-type}`	Summary	Count of all successful user self-service registrations by registration type and provider.
`idm_selfservice_user_registration{reg_type=registration-type}`	Summary	Count of all successful user self-service registrations by registration type.
`idm_sync_create_object_seconds`	Timer	Rate of requests to create an object on the target, and the time taken to perform this operation.
`idm_sync_delete_target_seconds`	Timer	Rate of requests to delete an object on the target, and the time taken to perform this operation.
`idm_sync_objectmapping_seconds{mapping_name=mapping-name}`	Timer	Rate of configurations applied to a mapping.
`idm_sync_queue_acquire{mapping_name=mapping-name, action=action}`	Timer	Rate of acquisition of queued synchronization events from the queue.
`idm_sync_queue_discard{mapping_name=mapping-name, action=action}`	Timer	Rate of deletion of synchronization events from the queue.
`idm_sync_queue_execution{mapping_name=mapping-name, action=action}`	Timer	Rate at which queued synchronization operations are executed.
`idm_sync_queue_failed{mapping_name=mapping-name, action=action}`	Summary	Number of queued synchronization operations that failed.
`idm_sync_queue_poll_pending_events{mapping_name=mapping-name}`	Timer	The latency involved in polling for synchronization events.
`idm_sync_queue_precondition_failed{mapping_name=mapping-name, action=action}`	Summary	Number of queued synchronization events that were acquired by another node in the cluster.

Prometheus Metric Name	Type	Description
`idm_sync_queue_rejected_executions{mapping_name=mapping-name, action=action}`	Summary	Number of queued synchronization events that were rejected because the backing thread-pool queue was at full capacity and the thread-pool had already allocated its maximum configured number of threads.
`idm_sync_queue_release_for_retry{mapping_name=mapping-name, action=action}`	Timer	Times the release of queued synchronization events after a failure and before exceeding the retry count.
`idm_sync_queue_release{mapping_name=mapping-name, action=action}`	Timer	Rate at which queued synchronization events are released.
`idm_sync_queue_submit{mapping_name=mapping-name, action=action}`	Timer	Rate of insertion of synchronization events into the queue.
`idm_sync_raw_read_object_seconds`	Timer	Rate of reads of an object.
`idm_sync_source_assess_situation_seconds`	Timer	Rate of assessments of a synchronization situation.
`idm_sync_source_correlate_target_seconds`	Timer	Rate of correlations between a target and a given source, and time taken to perform this operation.
`idm_sync_source_determine_action_seconds`	Timer	Rate of determinations done on a synchronization action based on its current situation.
`idm_sync_source_perform_action_seconds`	Timer	Rate of completions of an action performed on a synchronization operation.
`idm_sync_target_assess_situation_seconds`	Timer	Rate of assessments of a target situation.
`idm_sync_target_determine_action_seconds`	Timer	Rate of determinations done on a target action based on its current situation.

Prometheus Metric Name	Type	Description
`idm_sync_target_perform_action_seconds`	Timer	Rate of completions of an action performed on a target sync operation.
`idm_sync_update_target_seconds`	Timer	Rate of requests to update an object on the target, and the time taken to perform this operation.
`idm_user_login{user_type=user-type}`	Summary	Count of all successful logins by user type.
`idm_user_login_total{provider=provider,user_type=user-type}`	Summary	Count of all successful logins by user type and provider.
`idm_virtual_properties_from_relationships{virtual_properties=calculated-virtual-properties, traversal_depthX=traversal-origin-resource-collection and traversal relationship,not_found}`	Summary	Number of 404 responses encountered when querying the `resource_collection` / `relationship_field` specified in the traversal_depthX tag for the most recent X. X corresponds to the relationship field sequence.
`idm_virtual_properties_from_relationships{virtual_properties=calculated-virtual-properties, traversal_depthX=traversal-origin-resource-collection and traversal relationship,unsatisfied_temp_constraint}`	Summary	Number of edges skipped due to an unsatisfied temporal constraint on either the edge or the referred-to vertex. Encountered when querying the resource collection and relationship field at the traversal_depthX tag for the most recent X. X corresponds to the relationship field sequence.

Prometheus Metric Name	Type	Description
`idm_virtual_properties_from_relationships{virtual_properties=calculated-virtual-properties, traversal_depthX=traversal-origin-resource-collection and traversal relationship}`	Timer	Time spent traversing relationship fields to calculate the specified virtual properties. The managed objects linked to by the traversal relationship fields define a tree, whose root is the virtual property host. This object tree is traversed depth-first, with the traversal_depthX corresponding to the latency involved with each relationship traversal. Traversal_depth0 corresponds to the first relationship field traversed. Because the tree is traversed depth-first, traversal_depthX will subsume all the traversal latencies for all traversal_depth Y, where Y>X. X corresponds to the relationship field sequence.

Prometheus JVM metrics available in IDM

> **Note**
>
> These metrics depend on the JVM version and configuration. In particular, garbage-collector-related metrics depend on the garbage collector that the server uses. The garbage-collector metric names are unstable, and can change even in a minor JVM release.

Prometheus Metric Name	Type	Unit	Description
`idm_jvm_available_cpus`	Gauge	Count	Number of processors available to the JVM. For more information, refer to Runtime.
`idm_jvm_class_loading_loaded`	Gauge	Count	Number of classes loaded since the Java virtual machine started. For more information, refer to ClassLoadingMXBean.
`idm_jvm_class_loading_unloaded`	Gauge	Count	Number of classes unloaded since the Java virtual machine started. For more information, refer to ClassLoadingMXBean.

Prometheus Metric Name	Type	Unit	Description
`idm_jvm_free_used_memory_bytes`	Gauge	Bytes	For more information, refer to Runtime.
`idm_jvm_garbage_collector_g1_old_generation_count`	Gauge	Count	For each garbage collector in the JVM. For more information, refer to GarbageCollectorMXBean.
`idm_jvm_garbage_collector_g1_old_generation_time`	Gauge	Milliseconds	
`idm_jvm_garbage_collector_g1_young_generation_count`	Gauge	Count	
`idm_jvm_garbage_collector_g1_young_generation_time`	Gauge	Milliseconds	
`idm_jvm_max_memory_bytes`	Gauge	Bytes	For more information, refer to Runtime.
`idm_jvm_memory_usage_heap_committed`	Gauge	Bytes	Amount of heap memory committed for the JVM to use. For more information, refer to MemoryMXBean.
`idm_jvm_memory_usage_heap_init`	Gauge	Bytes	
`idm_jvm_memory_usage_heap_max`	Gauge	Bytes	Maximum amount of heap memory available to the JVM.
`idm_jvm_memory_usage_heap_usage`	Gauge	Bytes	
`idm_jvm_memory_usage_heap_used`	Gauge	Bytes	Amount of heap memory used by the JVM.
`idm_jvm_memory_usage_non_heap_committed`	Gauge	Bytes	Amount of non-heap memory committed for the JVM to use.
`idm_jvm_memory_usage_non_heap_init`	Gauge	Bytes	Amount of non-heap memory the JVM initially requested from the operating system.
`idm_jvm_memory_usage_non_heap_max`	Gauge	Bytes	Maximum amount of non-heap memory available to the JVM.
`idm_jvm_memory_usage_non_heap_usage`	Gauge	Bytes	
`idm_jvm_memory_usage_non_heap_used`	Gauge	Bytes	Amount of non-heap memory used by the JVM.

Prometheus Metric Name	Type	Unit	Description
`idm_jvm_memory_usage_pools_codeheap__non_nmethods__committed`	Gauge	Bytes	For each pool. For more information, refer to MemoryPoolMXBean.
`idm_jvm_memory_usage_pools_codeheap__non_nmethods__init`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__non_nmethods__max`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__non_nmethods__usage`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__non_nmethods__used`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__non_profiled_nmethods__committed`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__non_profiled_nmethods__init`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__non_profiled_nmethods__max`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__non_profiled_nmethods__usage`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__non_profiled_nmethods__used`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__profiled_nmethods__committed`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__profiled_nmethods__init`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__profiled_nmethods__max`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__profiled_nmethods__usage`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_codeheap__profiled_nmethods__used`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_compressed_class_space_committed`	Gauge	Bytes	

Prometheus Metric Name	Type	Unit	Description
`idm_jvm_memory_usage_pools_compressed_class_space_init`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_compressed_class_space_max`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_compressed_class_space_usage`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_compressed_class_space_used`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_eden_space_committed`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_eden_space_init`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_eden_space_max`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_eden_space_usage`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_eden_space_used`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_eden_space_used_after_gc`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_old_gen_committed`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_old_gen_init`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_old_gen_max`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_old_gen_usage`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_old_gen_used`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_old_gen_used_after_gc`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_survivor_space_committed`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_survivor_space_init`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_survivor_space_max`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_survivor_space_usage`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_g1_survivor_space_used`	Gauge	Bytes	

Prometheus Metric Name	Type	Unit	Description
`idm_jvm_memory_usage_pools_g1_survivor_space_used_after_gc`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_metaspace_committed`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_metaspace_init`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_metaspace_max`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_metaspace_usage`	Gauge	Bytes	
`idm_jvm_memory_usage_pools_metaspace_used`	Gauge	Bytes	
`idm_jvm_memory_usage_total_committed`	Gauge	Bytes	Amount of memory that is committed for the JVM to use. For more information, refer to MemoryMXBean.
`idm_jvm_memory_usage_total_init`	Gauge	Bytes	
`idm_jvm_memory_usage_total_max`	Gauge	Bytes	
`idm_jvm_memory_usage_total_used`	Gauge	Bytes	
`idm_jvm_thread_state_blocked_count`	Gauge	Count	For more information, refer to ThreadMXBean.
`idm_jvm_thread_state_count`	Gauge	Count	Number of live threads including both daemon and non-daemon threads.
`idm_jvm_thread_state_daemon_count`	Gauge	Count	Number of live daemon threads.
`idm_jvm_thread_state_new_count`	Gauge	Count	Number of threads in the `NEW` state.
`idm_jvm_thread_state_runnable_count`	Gauge	Count	Number of threads in the `RUNNABLE` state.
`idm_jvm_thread_state_terminated_count`	Gauge	Count	Number of threads in the `TERMINATED` state.
`idm_jvm_thread_state_timed_waiting_count`	Gauge	Count	Number of threads in the `TIMED_WAITING` state.
`idm_jvm_thread_state_waiting_count`	Gauge	Count	Number of threads in the `WAITING` state.

Prometheus Metric Name	Type	Unit	Description
`idm_jvm_used_memory_bytes`	Gauge	Bytes	For more information, refer to totalMemory().

Prometheus workflow metrics available in IDM

Prometheus Metric Name	Type	Description
`idm_workflow_execution_action_seconds{action="message"}`	Timer	Time spent invoking a message event.
`idm_workflow_execution_action_seconds{action="signal"}`	Timer	Time spent invoking a signal event.
`idm_workflow_execution_action_seconds{action="trigger"}`	Timer	Time spent triggering an execution.
`idm_workflow_execution_query_seconds`	Timer	Time spent querying executions.
`idm_workflow_job_action_seconds{action="execute"}`	Timer	Time spent forcing synchronous execution of a job.
`idm_workflow_job_action_seconds{action="stacktrace"}`	Timer	Time spent displaying the stacktrace for a job that triggered an exception.
`idm_workflow_job_delete_seconds`	Timer	Time spent deleting a job.
`idm_workflow_job_query_seconds`	Timer	Time spent querying jobs.
`idm_workflow_job_read_seconds`	Timer	Time spent reading a single job.
`idm_workflow_jobdeadletter_action_seconds{action="execute"}`	Timer	Time spent to execute dead-letter job.
`idm_workflow_jobdeadletter_action_seconds{action="stacktrace"}`	Timer	Time spent to retrieve the stacktrace for a dead-letter job.
`idm_workflow_jobdeadletter_delete_seconds`	Timer	Time spent to delete a dead letter job.
`idm_workflow_jobdeadletter_query_seconds`	Timer	Time spent to query dead letter jobs.
`idm_workflow_jobdeadletter_read_seconds`	Timer	Time spent to read a dead letter job.
`idm_workflow_model_action_seconds{action="deploy"}`	Timer	Time spent to deploy a model.
`idm_workflow_model_action_seconds{action="list_deployments"}`	Timer	Time spent to list model deployments.
`idm_workflow_model_action_seconds{action="validate_bpmn"}`	Timer	Time spent to validate BPMN content.
`idm_workflow_model_create_seconds`	Timer	Time spent to create a model.

Prometheus Metric Name	Type	Description
`idm_workflow_model_delete_seconds`	Timer	Time spent to delete a model.
`idm_workflow_model_query_seconds`	Timer	Time spent to query models.
`idm_workflow_model_read_seconds`	Timer	Time spent to read a model.
`idm_workflow_model_update_seconds`	Timer	Time spent to update a model.
`idm_workflow_processdefinition_delete_seconds`	Timer	Time spent to delete a process definition.
`idm_workflow_processdefinition_query_seconds`	Timer	Time spent to query process definitions.
`idm_workflow_processdefinition_read_seconds`	Timer	Time spent to read a process definition.
`idm_workflow_processinstance_action_seconds{action="migrate"}`	Timer	Time spent to migrate a process instance.
`idm_workflow_processinstance_action_seconds{action="validateMigration"}`	Timer	Time spent to validate a migration of a process instance.
`idm_workflow_processinstance_create_seconds`	Timer	Time spent to create a process instance.
`idm_workflow_processinstance_delete_seconds`	Timer	Time spent to delete a process instance.
`idm_workflow_processinstance_query_seconds`	Timer	Time spent to query process instances.
`idm_workflow_processinstance_read_seconds`	Timer	Time spent to read a process instance.
`idm_workflow_taskdefinition_query_seconds`	Timer	Time spent to query task definitions.
`idm_workflow_taskdefinition_read_seconds`	Timer	Time spent to read a task definition.
`idm_workflow_taskinstance_action_seconds{action="complete"}`	Timer	Time spent to complete a task instance.
`idm_workflow_taskinstance_query_seconds`	Timer	Time spent to query task instances.
`idm_workflow_taskinstance_read_seconds`	Timer	Time spent to read a task instance.
`idm_workflow_taskinstance_update_seconds`	Timer	Time spent to update a task instance.

REST API reference

> *Guide to creating and managing objects in ForgeRock® Identity Management.*

This reference describes the ForgeRock Common REST API. refer to Common REST and IDM for information specific to the IDM implementation of Common REST.

Quick Start

Start Here

Learn about the Common REST interface in the ForgeRock Platform and the specifics of REST in IDM.

API Explorer

Access the online IDM REST API reference through the admin UI.

REST API Structure

Understand RESTful syntax with respect to the IDM REST API.

REST Endpoints

Discover the REST endpoints IDM exposes.

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

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

ForgeRock Common REST

ForgeRock® Common REST is a common REST API framework. It works across the ForgeRock platform to provide common ways to access web resources and collections of resources. Adapt the examples in this section to your resources and deployment.

> **Note**
>
> This section describes the full Common REST framework. Some platform component products do not implement all Common REST behaviors exactly as described in this section. For details, refer to the product-specific examples and reference information in other sections of this documentation set.

Common REST Resources

Servers generally return JSON-format resources, though resource formats can depend on the implementation.

Resources in collections can be found by their unique identifiers (IDs). IDs are exposed in the resource URIs. For example, if a server has a user collection under `/users`, then you can access a user at `/users/user-id`. The ID is also the value of the `_id` field of the resource.

Resources are versioned using revision numbers. A revision is specified in the resource's `_rev` field. Revisions make it possible to figure out whether to apply changes without resource locking and without distributed transactions.

Common REST Verbs

The Common REST APIs use the following verbs, sometimes referred to collectively as `CRUDPAQ`. For details and HTTP-based examples of each, follow the links to the sections for each verb.

Create

Add a new resource.

This verb maps to HTTP PUT or HTTP POST.

For details, refer to Create.

Read

Retrieve a single resource.

This verb maps to HTTP GET.

For details, refer to Read.

Update

Replace an existing resource.

This verb maps to HTTP PUT.

For details, refer to Update.

Delete

Remove an existing resource.

This verb maps to HTTP DELETE.

For details, refer to Delete.

Patch

Modify part of an existing resource.

This verb maps to HTTP PATCH.

For details, refer to Patch.

Action

Perform a predefined action.

This verb maps to HTTP POST.

For details, refer to Action.

Query

Search a collection of resources.

This verb maps to HTTP GET.

For details, refer to Query.

Common REST Parameters

Common REST reserved query string parameter names start with an underscore (`_`).

Reserved query string parameters include, but are not limited to, the following names:

`_action`
`_api`
`_crestapi`
`_fields`
`_mimeType`
`_pageSize`
`_pagedResultsCookie`
`_pagedResultsOffset`
`_prettyPrint`
`_queryExpression`
`_queryFilter`
`_queryId`
`_sortKeys`
`_totalPagedResultsPolicy`

> **Note**
>
> Some parameter values are not safe for URLs, so URL-encode parameter values as necessary.

Continue reading for details about how to use each parameter.

Common REST Extension Points

The *action* verb is the main vehicle for extensions. For example, to create a new user with HTTP POST rather than HTTP PUT, you might use `/users?_action=create`. A server can define additional actions. For example, `/tasks/1?_action=cancel`.

A server can define *stored queries* to call by ID. For example, `/groups?_queryId=hasDeletedMembers`. Stored queries can call for additional parameters. The parameters are also passed in the query string. Which parameters are valid depends on the stored query.

Common REST API Documentation

Common REST APIs often depend at least in part on runtime configuration. Many Common REST endpoints therefore serve *API descriptors* at runtime. An API descriptor documents the actual API as it is configured.

Use the following query string parameters to retrieve API descriptors:

`_api`

Serves an API descriptor that complies with the OpenAPI specification.

This API descriptor represents the API accessible over HTTP. It's suitable for use with popular tools, such as Swagger UI.

> **Note**
>
> Requests made with the `_api` query string parameter require authorization. Grant access by creating a custom rule in `access.json`. The following example authorizes users with the `openidm-authorized` role:
>
> ```
> {
>     "pattern" : "managed/user",
>     "roles" : "internal/role/openidm-authorized",
>     "methods" : "read",
>     "customAuthz" : "checkIfApiRequest()"
> }
> ```

`_crestapi`

Serves a native Common REST API descriptor.

This API descriptor provides a compact representation that is not dependent on the transport protocol. It requires a client that understands Common REST, as it omits many Common REST defaults.

> **Note**
>
> Consider limiting access to API descriptors in production environments in order to avoid unnecessary traffic. To provide documentation in production environments, refer to Publish OpenAPI Documentation instead.

Publish OpenAPI Documentation

In production systems, developers expect stable, well-documented APIs. Rather than retrieving API descriptors at runtime through Common REST, prepare final versions, and publish them alongside the software in production.

Use the OpenAPI-compliant descriptors to provide API reference documentation for your developers as described in the following steps:

1. Configure the software to produce production-ready APIs.

   In other words, the software should be configured as in production so that the APIs are identical to what developers refer to in production.

2. Retrieve the OpenAPI-compliant descriptor.

   The following command saves the descriptor to a file, `myapi.json`:

   ```
   curl -o myapi.json endpoint?_api
   ```

3. If necessary, edit the descriptor.

   For example, you might want to add security definitions to describe how the API is protected.

   If you make any changes, then also consider using a source control system to manage your versions of the API descriptor.

4. Publish the descriptor using a tool, such as Swagger UI.

   You can customize Swagger UI for your organization as described in the documentation for the tool.

Create

There are two ways to create a resource, either with an HTTP POST or with an HTTP PUT.

To create a resource using POST, perform an HTTP POST with the query string parameter `_action=create` and the JSON resource as a payload. The server creates the identifier if not specified:

```
POST /users?_action=create HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
{ JSON resource }
```

To create a resource using PUT, perform an HTTP PUT including the case-sensitive identifier for the resource in the URL path, and the JSON resource as a payload. Optionally, include the `If-None-Match: *` header to prevent overwriting an existing object:

```
PUT /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-None-Match: *
{ JSON resource }
```

The `_id` and content of the resource depend on the server implementation. The server is not required to use the `_id` that the client provides. The server response to the create request indicates the resource location as the value of the `Location` header.

If you include the `If-None-Match` header, its value must be `*`. In this case, the request creates the object if it does not exist, and fails if the object does exist. If you include the `If-None-Match` header with any value other than `*`, the server returns an HTTP 400 Bad Request error. For example, creating an object with `If-None-Match: revision` returns a bad request error. If you do not include `If-None-Match: *`, the request creates the object if it does not exist, and *updates* the object if it does exist.

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Read

To retrieve a single resource, perform an HTTP GET on the resource by its case-sensitive identifier (`_id`) and accept a JSON response:

```
GET /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
```

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

`_mimeType=mime-type`

Some resources have fields whose values are multi-media resources such as a profile photo for example.

If the feature is enabled for the endpoint, you can read a single field that is a multi-media resource by specifying the field and mime-type.

In this case, the content type of the field value returned matches the mime-type that you specify, and the body of the response is the multi-media resource.

The `Accept` header is not used in this case. For example, `Accept: image/png` does not work. Use the `_mimeType` query string parameter instead.

Update

To update a resource, perform an HTTP PUT including the case-sensitive identifier (`_id`) as the final element of the path to the resource, and the JSON resource as the payload. Use the `If-Match: _rev` header to check that you are actually updating the version you modified. Use `If-Match: *` if the version does not matter.

```
PUT /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-Match: _rev
{ JSON resource }
```

When updating a resource, include all the attributes to be retained. Omitting an attribute in the resource amounts to deleting the attribute unless it is not under the control of your application. Attributes not under the control of your application include private and read-only attributes. In addition, virtual attributes and relationship references might not be under the control of your application.

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}` , `parent/child` refers to the `"child":"value"` .

If the `field` is left blank, the server returns all default values.

Delete

To delete a single resource, perform an HTTP DELETE by its case-sensitive identifier (`_id`) and accept a JSON response:

```
DELETE /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
```

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}` , `parent/child` refers to the `"child":"value"` .

If the `field` is left blank, the server returns all default values.

Patch

To patch a resource, send an HTTP PATCH request with the following parameters:

- `operation`
- `field`
- `value`
- `from` (optional with copy and move operations)

You can include these parameters in the payload for a PATCH request, or in a JSON PATCH file. If successful, you'll refer to a JSON response similar to:

```
PATCH /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-Match: _rev
{ JSON array of patch operations }
```

PATCH operations apply to three types of targets:

- **single-valued**, such as an object, string, boolean, or number.

- **list semantics array**, where the elements are ordered, and duplicates are allowed.
- **set semantics array**, where the elements are not ordered, and duplicates are not allowed.

ForgeRock PATCH supports several different `operations`. The following sections show each of these operations, along with options for the `field` and `value`:

Patch Operation: Add

The `add` operation ensures that the target field contains the value provided, creating parent fields as necessary.

If the target field is single-valued, then the value you include in the PATCH replaces the value of the target. Examples of a single-valued field include: object, string, boolean, or number.

An `add` operation has different results on two standard types of arrays:

- **List semantic arrays**: you can run any of these `add` operations on that type of array:
  - If you `add` an array of values, the PATCH operation appends it to the existing list of values.
  - If you `add` a single value, specify an ordinal element in the target array, or use the `{-}` special index to add that value to the end of the list.
- **Set semantic arrays**: The value included in the patch is merged with the existing set of values.

As an example, start with the following list semantic array resource:

```
{
    "fruits" : [ "orange", "apple" ]
}
```

The following add operation includes the pineapple to the end of the list of fruits, as indicated by the `-` at the end of the `fruits` array.

```
{
    "operation" : "add",
    "field" : "/fruits/-",
    "value" : "pineapple"
}
```

The following is the resulting resource:

```
{
    "fruits" : [ "orange", "apple", "pineapple" ]
}
```

Note that you can add only one array element one at a time, as per the corresponding JSON Patch specification. If you add an array of elements, for example:

```
{
    "operation" : "add",
    "field" : "/fruits/-",
    "value" : ["pineapple", "mango"]
}
```

The resulting resource would have the following invalid JSON structure:

```
{
    "fruits" : [ "orange", "apple", ["pineapple", "mango"]]
}
```

Patch Operation: Copy

The copy operation takes one or more existing values from the source field. It then adds those same values on the target field. Once the values are known, it is equivalent to performing an `add` operation on the target field.

The following `copy` operation takes the value from a field named `mail`, and then runs a `replace` operation on the target field, `another_mail`.

```
[
  {
     "operation":"copy",
     "from":"mail",
     "field":"another_mail"
  }
]
```

If the source field value and the target field value are configured as arrays, the result depends on whether the array has list semantics or set semantics, as described in Patch Operation: Add.

Patch Operation: Increment

The `increment` operation changes the value or values of the target field by the amount you specify. The value that you include must be one number, and may be positive or negative. The value of the target field must accept numbers. The following `increment` operation adds `1000` to the target value of `/user/payment`.

```
[
  {
    "operation" : "increment",
    "field" : "/user/payment",
    "value" : "1000"
  }
]
```

Since the `value` of the `increment` is a single number, arrays do not apply.

Patch Operation: Move

The move operation removes existing values on the source field. It then adds those same values on the target field. It is equivalent to performing a `remove` operation on the source, followed by an `add` operation with the same values, on the target.

The following `move` operation is equivalent to a `remove` operation on the source field, `surname`, followed by a `replace` operation on the target field value, `lastName`. If the target field does not exist, it is created.

```
[
  {
     "operation":"move",
     "from":"surname",
     "field":"lastName"
  }
]
```

To apply a `move` operation on an array, you need a compatible single-value, list semantic array, or set semantic array on both the source and the target. For details, refer to the criteria described in Patch Operation: Add.

Patch Operation: Remove

The `remove` operation ensures that the target field no longer contains the value provided. If the remove operation does not include a value, the operation removes the field. The following `remove` deletes the value of the `phoneNumber`, along with the field.

```
[
  {
    "operation" : "remove",
    "field" : "phoneNumber"
  }
]
```

If the object has more than one `phoneNumber`, those values are stored as an array.

A `remove` operation has different results on two standard types of arrays:

- **List semantic arrays**: A `remove` operation deletes the specified element in the array. For example, the following operation removes the first phone number, based on its array index (zero-based):

  ```
  [
     {
        "operation" : "remove",
        "field" : "/phoneNumber/0"
     }
  ]
  ```

- **Set semantic arrays**: The list of values included in a patch are removed from the existing array.

Patch Operation: Replace

The `replace` operation removes any existing value(s) of the targeted field, and replaces them with the provided value(s). It is essentially equivalent to a `remove`, followed by a `add` operation. If the arrays are used, the criteria is based on Patch Operation: Add. However, indexed updates are not allowed, even when the target is an array.

The following `replace` operation removes the existing `telephoneNumber` value for the user, and then adds the new value of `+1 408 555 9999`.

```
[
  {
    "operation" : "replace",
    "field" : "/telephoneNumber",
    "value" : "+1 408 555 9999"
  }
]
```

A PATCH replace operation on a list semantic array works in the same fashion as a PATCH remove operation. The following example demonstrates how the effect of both operations. Start with the following resource:

```
{
    "fruits" : [ "apple", "orange", "kiwi", "lime" ],
}
```

Apply the following operations on that resource:

```
[
  {
    "operation" : "remove",
    "field" : "/fruits/0",
    "value" : ""
  },
  {
    "operation" : "replace",
    "field" : "/fruits/1",
    "value" : "pineapple"
  }
]
```

The PATCH operations are applied sequentially. The `remove` operation removes the first member of that resource, based on its array index, (`fruits/0`), with the following result:

```
[
  {
    "fruits" : [ "orange", "kiwi", "lime" ],
  }
]
```

The second PATCH operation, a `replace`, is applied on the second member (`fruits/1`) of the intermediate resource, with the following result:

```
[
  {
    "fruits" : [ "orange", "pineapple", "lime" ],
  }
]
```

Patch Operation: Transform

The `transform` operation changes the value of a field based on a script or some other data transformation command. The following `transform` operation takes the value from the field named `/objects`, and applies the `something.js` script as shown:

```
[
  {
    "operation" : "transform",
    "field" : "/objects",
    "value" : {
      "script" : {
        "type" : "text/javascript",
        "file" : "something.js"
      }
    }
  }
]
```

Patch Operation Limitations

Some HTTP client libraries do not support the HTTP PATCH operation. Make sure that the library you use supports HTTP PATCH before using this REST operation.

For example, the Java Development Kit HTTP client does not support PATCH as a valid HTTP method. Instead, the method `HttpURLConnection.setRequestMethod("PATCH")` throws `ProtocolException`.

Parameters

You can use the following parameters. Other parameters might depend on the specific action implementation:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Action

Actions are a means of extending Common REST APIs and are defined by the resource provider, so the actions you can use depend on the implementation.

The standard action indicated by `_action=create` is described in Create.

Parameters

You can use the following parameters. Other parameters might depend on the specific action implementation:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Query

To query a resource collection (or resource container if you prefer to think of it that way), perform an HTTP GET and accept a JSON response, including at least a `_queryExpression`, `_queryFilter`, or `_queryId` parameter. These parameters cannot be used together:

```
GET /users?_queryFilter=true HTTP/1.1
Host: example.com
Accept: application/json
```

The server returns the result as a JSON object including a "results" array and other fields related to the query string parameters that you specify.

Parameters

You can use the following parameters:

`_queryFilter=filter-expression`

Query filters request that the server return entries that match the filter expression. You must URL-escape the filter expression.

The string representation is summarized as follows. Continue reading for additional explanation:

```
Expr           = OrExpr
OrExpr         = AndExpr ( 'or' AndExpr ) *
AndExpr        = NotExpr ( 'and' NotExpr ) *
NotExpr        = '!' PrimaryExpr | PrimaryExpr
PrimaryExpr    = '(' Expr ')' | ComparisonExpr | PresenceExpr | LiteralExpr
ComparisonExpr = Pointer OpName JsonValue
PresenceExpr   = Pointer 'pr'
LiteralExpr    = 'true' | 'false'
Pointer        = JSON pointer
OpName         = 'eq' |  # equal to
                 'co' |  # contains
                 'sw' |  # starts with
                 'lt' |  # less than
                 'le' |  # less than or equal to
                 'gt' |  # greater than
                 'ge' |  # greater than or equal to
                 STRING  # extended operator
JsonValue      = NUMBER | BOOLEAN | '"' UTF8STRING '"'
STRING         = ASCII string not containing white-space
UTF8STRING     = UTF-8 string possibly containing white-space
```

JsonValue components of filter expressions follow RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format. In particular, as described in section 7 of the RFC, the escape character in strings is the backslash character. For example, to match the identifier `test\`, use `_id eq 'test\\'`. In the JSON resource, the `\` is escaped the same way: `"_id":"test\\"`.

When using a query filter in a URL, be aware that the filter expression is part of a query string parameter. A query string parameter must be URL encoded as described in RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. For example, white space, double quotes (`"`), parentheses, and exclamation characters need URL encoding in HTTP query strings. The following rules apply to URL query components:

```
query       = *( pchar / "/" / "?" )
pchar       = unreserved / pct-encoded / sub-delims / ":" / "@"
unreserved  = ALPHA / DIGIT / "-" / "." / "_" / "~"
pct-encoded = "%" HEXDIG HEXDIG
sub-delims  = "!" / "$" / "&" / "'" / "(" / ")"
                  / "*" / "+" / "," / ";" / "="
```

`ALPHA`, `DIGIT`, and `HEXDIG` are core rules of RFC 5234: Augmented BNF for Syntax Specifications:

```
ALPHA       =  %x41-5A / %x61-7A   ; A-Z / a-z
DIGIT       =  %x30-39             ; 0-9
HEXDIG      =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
```

As a result, a backslash escape character in a JsonValue component is percent-encoded in the URL query string parameter as `%5C`. To encode the query filter expression `_id eq 'test\\'`, use `_id+eq'test%5C%5C'+`, for example.

A simple filter expression can represent a comparison, presence, or a literal value.

For comparison expressions use json-pointer comparator json-value, where the comparator is one of the following:

`eq` (equals)
`co` (contains)
`sw` (starts with)
`lt` (less than)
`le` (less than or equal to)
`gt` (greater than)
`ge` (greater than or equal to)

For presence, use json-pointer pr to match resources where:

- The JSON pointer is present.
- The value it points to is not `null`.

Literal values include true (match anything) and false (match nothing).

Complex expressions employ `and`, `or`, and `!` (not), with parentheses, `(expression)`, to group expressions.

`_queryId=identifier`

Specify a query by its identifier.

Specific queries can take their own query string parameter arguments, which depend on the implementation.

`_pagedResultsCookie=string`

The string is an opaque cookie used by the server to keep track of the position in the search results. The server returns the cookie in the JSON response as the value of `pagedResultsCookie`.

In the request `_pageSize` must also be set and non-zero. You receive the cookie value from the provider on the first request, and then supply the cookie value in subsequent requests until the server returns a `null` cookie, meaning that the final page of results has been returned.

The `_pagedResultsCookie` parameter is supported when used with the `_queryFilter` parameter. The `_pagedResultsCookie` parameter is not guaranteed to work when used with the `_queryExpression` and `_queryId` parameters.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and not to be used together.

`_pagedResultsOffset=integer`

When `_pageSize` is non-zero, use this as an index in the result set indicating the first page to return.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and not to be used together.

`_pageSize=integer`

Return query results in pages of this size. After the initial request, use `_pagedResultsCookie` or `_pageResultsOffset` to page through the results.

`_totalPagedResultsPolicy=string`

When a `_pageSize` is specified, and non-zero, the server calculates the "totalPagedResults", in accordance with the `totalPagedResultsPolicy`, and provides the value as part of the response. The "totalPagedResults" is either an estimate of the total number of paged results (`_totalPagedResultsPolicy=ESTIMATE`), or the exact total result count (`_totalPagedResultsPolicy=EXACT`). If no count policy is specified in the query, or if `_totalPagedResultsPolicy=NONE`, result counting is disabled, and the server returns value of -1 for "totalPagedResults".

`_sortKeys=[-][.replaceable]##field##[,[-]field...]`

Sort the resources returned based on the specified field(s), either in `+` (ascending, default) order, or in `-` (descending) order.

Because ascending order is the default, including the `` ` character in the query is unnecessary. If you do include the ``, it must be URL-encoded as `%2B` ``, for example:

```
http://localhost:8080/api/users?_prettyPrint=true&_queryFilter=true&_sortKeys=%2Bname/givenName
```

The `_sortKeys` parameter is not supported for predefined queries (`_queryId`).

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in each element of the "results" array in the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

HTTP status codes

When working with a Common REST API over HTTP, client applications should expect at least the following HTTP status codes. Not all servers necessarily return all status codes identified here:

200 OK

The request was successful and a resource returned, depending on the request.

201 Created

The request succeeded and the resource was created.

204 No Content

The action request succeeded, and there was no content to return.

304 Not Modified

The read request included an `If-None-Match` header, and the value of the header matched the revision value of the resource.

400 Bad Request

The request was malformed.

401 Unauthorized

The request requires user authentication.

403 Forbidden

Access was forbidden during an operation on a resource.

404 Not Found

The specified resource could not be found, perhaps because it does not exist.

405 Method Not Allowed

The HTTP method is not allowed for the requested resource.

406 Not Acceptable

The request contains parameters that are not acceptable, such as a resource or protocol version that is not available.

409 Conflict

The request would have resulted in a conflict with the current state of the resource.

410 Gone

The requested resource is no longer available, and will not become available again. This can happen when resources expire for example.

412 Precondition Failed

The resource's current version does not match the version provided.

415 Unsupported Media Type

The request is in a format not supported by the requested resource for the requested method.

428 Precondition Required

The resource requires a version, but no version was supplied in the request.

500 Internal Server Error

The server encountered an unexpected condition that prevented it from fulfilling the request.

501 Not Implemented

The resource does not support the functionality required to fulfill the request.

503 Service Unavailable

The requested resource was temporarily unavailable. The service may have been disabled, for example.

REST and IDM

Representational State Transfer (REST) is a software architecture style for exposing resources, using the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed.

IDM provides a RESTful API for accessing managed objects, system objects, workflows, and the system configuration.

Common REST and IDM

IDM implements the Common REST API as described in the previous section, with the exception of the following elements:

- IDM provides limited support for the in expression clause. You can use this clause for queries on singleton string properties, not arrays. `in` query expressions are not supported through the admin UI.
- The PATCH `transform` action is supported only on the `config` endpoint. Note that this is an optional action and not implemented everywhere across the ForgeRock Identity Platform.
- Common REST supports PATCH operations by list element index, as shown in the example in Patch Operation: Remove. IDM *does not support* PATCH by list element index. So, for PATCH operations, you cannot use an ordinal when adding or removing list items.

  You can add an item using the special hyphen index, which designates that the element should be added to the end of the list. To remove specific items from a list, you must specify the *value* to be removed, for example:

  ```
  [
      {
          "operation" : "remove",
          "field" : "/phoneNumber/",
          "value" : "202-555-0185"
      }
  ]
  ```

  Note

  When you remove items in this way, if the list contains two or more items with the same value, they are all removed.

- If `_fields` is left blank (null), the server returns all default values. In IDM, this excludes relationships and virtual fields. To include these fields in the output, add `"returnByDefault" : true` in the applicable schema.

  IDM also implements wild-card (`*`) handling with the `_fields` parameter. So, a value of `_fields=*_ref` will return all relationship fields associated with an object. A value of `_fields=*_ref/*` will return all the fields within each relationship.

- IDM does not implement the `ESTIMATE` total paged results policy. The `totalPagedResults` is either the exact total result count (`_totalPagedResultsPolicy=EXACT`) or result counting is disabled (`_totalPagedResultsPolicy=NONE`). For more information, refer to Page Query Results.

REST API Explorer

IDM includes an API Explorer, an implementation of the OpenAPI Initiative Specification, also known as Swagger.

The API Explorer covers most endpoints provided with a default IDM installation.

Each endpoint lists supported HTTP methods, such as POST and GET. When custom actions are available, the API Explorer lists them as:

```
HTTP Method /path/to/endpoint?_action=something
```

Example

To see the API Explorer in action, follow along with this procedure:

1. To access the API Explorer, log in to the admin UI, click the question mark button ? in the upper right corner, and select **</> API Explorer**.

   > **Note**
   >
   > If the API Explorer does not display, you might need to enable it in your `resolver/boot.properties` file by setting the `openidm.apidescriptor.enabled` property to `true` .

2. Expand the **User v1.0** endpoint node, and click **GET /openidm/managed/user1.0_query_id_query-all**.

3. Click **Try it out!**, and then click **Execute**.

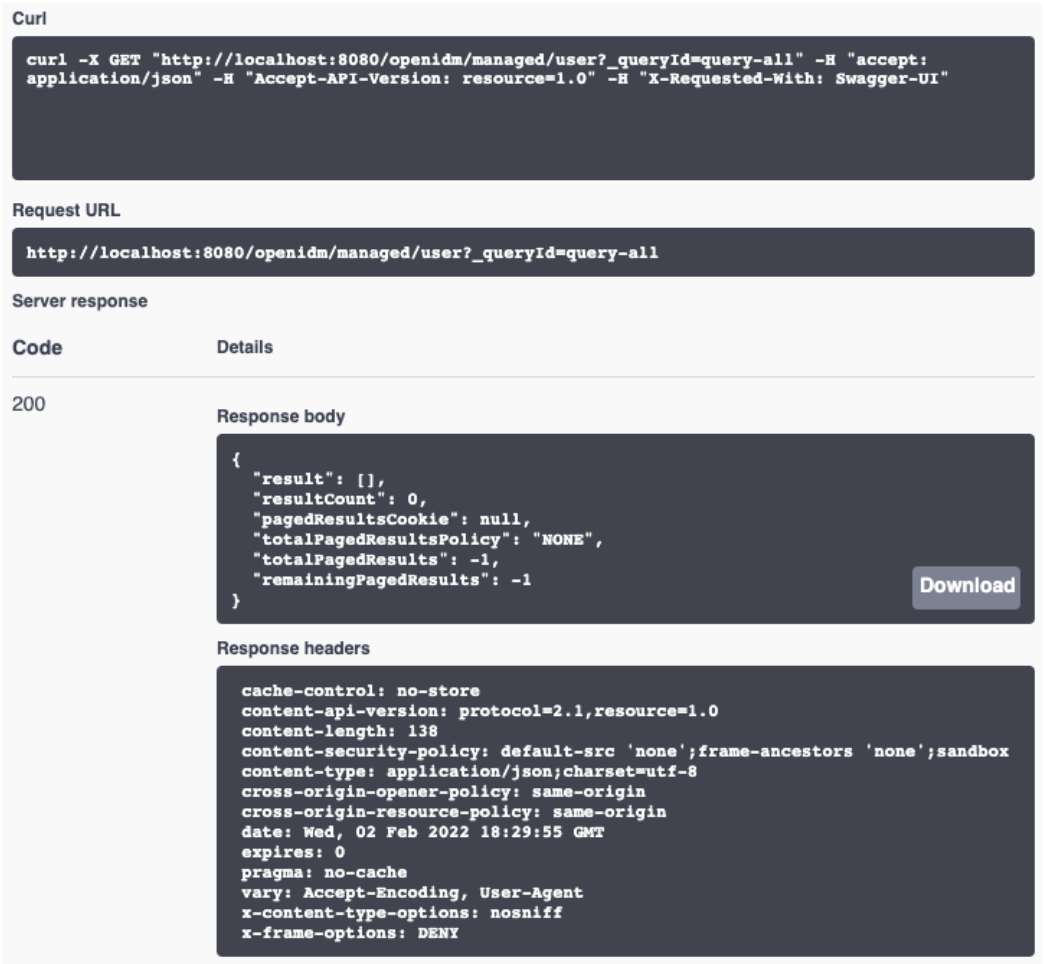
   The output includes:

   - The REST call, in the form of the `curl` command.
   - The request URL, which specifies the endpoint and associated parameters.

- The response body, which contains the data that you requested.
- The HTTP response code; if everything works, this should be `200`.

> **Tip**
>
> If you refer to a `401 Access Denied` code in the response body, your session may have timed out, and you'll have to log in to the admin UI again.

- Response headers.



For details on common ForgeRock REST parameters, refer to ForgeRock Common REST.

You'll refer to examples of REST calls throughout this documentation set. You can try these calls with the API Explorer.

You can also generate an OpenAPI-compliant descriptor of the REST API to provide API reference documentation specific to your deployment. The following command saves the API descriptor of the managed/user endpoint to a file named `my-openidm-api.json`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
--output "my-openidm-api.json" \
"http://localhost:8080/openidm/managed/user?_api"
```

For information about publishing reference documentation using the API descriptor, refer to Publish OpenAPI Documentation.

REST API versioning

ForgeRock REST API features are assigned version numbers. Providing version numbers in the REST API helps ensure compatibility between releases. The version number of a feature increases when ForgeRock introduces a change that is not backwards-compatible, and that affects clients that use the feature.

If there is more than one version of the API, you must select the version by setting a version header that specifies which version of the *resource* is requested. To ensure that your clients are always compatible with a newer IDM version, you should always include resource versions in your REST calls.

Specify the API version in REST calls

HTTP requests can optionally include the `Accept-API-Version` header with the value of the resource version, such as `resource=2.0`. If no `Accept-API-Version` header is included, the *latest* resource version is invoked by the HTTP request.

The following call requests version `2.0` of the specified resource:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0" \
--request POST \
--data '{
  "url":"https://www.forgerock.com/favicon.ico",
  "method":"GET"
}' \
"http://localhost:8080/openidm/external/rest?_action=call"
```

If `Accept-API-Version` contains an invalid version, IDM returns the following error:

```
{
  "code": 404,
  "reason": "Not Found",
  "message": "Resource&#39;&#39; not found"
}
```

Specify the API Version in Scripts

You can specify a resource version in scripts using the fourth (*additional parameters*) argument. If present, the `Accept-API-Version` parameter is applied to the actual REST request. Any other parameters are set as Additional Parameters on the request.

The following examples request specific resource versions:

REST with Inline Javascript

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "type":"text/javascript",
  "source":"openidm.action(\"external/rest\", \"call\", {\"url\": \"https://www.forgerock.com/
favicon.ico\", \"method\": \"GET\"}, {\"Accept-API-Version\": \"resource=1.0\"});"
}' \
"http://localhost:8080/openidm/script?_action=eval"
```

Standalone Javascript

```
openidm.action("external/rest", "call",
  {"url": "https://www.forgerock.com/favicon.ico", "method": "GET"},
  {"Accept-API-Version": "resource=1.0"});
```

API Version Header Warnings

IDM can log warnings when API version headers are not specified. Additionally, you can enable warnings when *scripts* don't specify API versions. Warnings are disabled by default. To enable this feature, add one or more of the following to your project's `resolver/boot.properties` file:

`openidm.apiVersion.warning.enabled=true`

- A message will be logged once per resource path, at the `info` level. For example:

  ```
  INFO: Accept-API-Version header missing from external request (authentication);
  transactionId=e017258a-8bac-4507-9575-78a41152e479-1929
  ```

- The HTTP response will apply a warning header. For example:

  ```
  Warning: 100 CREST "Accept-API-Version should be included in the request."
  ```

`openidm.apiVersion.warning.includeScripts=true`

> **Note**
>
> This setting requires `openidm.apiVersion.warning.enabled=true`.

- A message will be logged once per resource path *and* script-name pair, at the `info` level.

  Example script file log entry:

  ```
  [127] Sep 22, 2021 4:08:15.162 AM
  org.forgerock.openidm.servlet.internal.ResourceApiVersionFilterRegistration
  logOnceForScriptRequest
  INFO: Accept-API-Version header missing from script (policyFilter.js) request: policy
  ```

  Example inline script log entry:

  ```
  INFO: Accept-API-Version header missing from script
  (d6fc81179beaca37094a23c2fcd00aaf54bb3ef9:router:onRequest) request (config)
  ...
  INFO: Accept-API-Version header missing from script (policy.js) request (managed/user)
  ```

Filter Resource Path Warnings

To filter which resource paths are logged, edit the `logFilterResourcePaths` array located in the `conf/apiVersion.json` file. You can also modify the configuration over REST:

1. Get the current configuration:

   ```
   curl \
   --header "X-OpenIDM-Username: openidm-admin" \
   --header "X-OpenIDM-Password: openidm-admin" \
   --header "Accept-API-Version: resource=1.0" \
   --request GET \
   "http://localhost:8080/openidm/config/apiVersion"
   ```

```
{
    "warning" : {
        "enabled" : {
            "$bool" : "&{openidm.apiVersion.warning.enabled|false}"
        },
        "includeScripts" : {
            "$bool" : "&{openidm.apiVersion.warning.includeScripts|false}"
        },
        "logFilterResourcePaths" : [
            "audit",
            "authentication",
            "cluster",
            "config",
            "consent",
            "csv",
            "external/rest",
            "identityProviders",
            "info",
            "internal",
            "internal/role",
            "internal/user",
            "internal/usermeta",
            "managed",
            "managed/assignment",
            "managed/organization",
            "managed/role",
            "managed/user",
            "notification",
            "policy",
            "privilege",
            "profile",
            "recon",
            "recon/assoc",
            "repo",
            "selfservice/kba",
            "selfservice/terms",
            "scheduler/job",
            "scheduler/trigger",
            "schema",
            "sync",
            "sync/mappings",
            "system",
            "taskscanner"
        ]
    }
}
```

2. Make changes, and replace the configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--request PUT \
--data '{
    "warning" : {
        "enabled" : {
            "$bool" : "&{openidm.apiVersion.warning.enabled|false}"
        },
        "includeScripts" : {
            "$bool" : "&{openidm.apiVersion.warning.includeScripts|false}"
        },
        "logFilterResourcePaths" : [ <Insert modified resourcePaths here>
        ]
    }
}' \
"http://localhost:8080/openidm/config/apiVersion"
```

REST API structure

URI scheme

The URI scheme for accessing a managed object follows this convention, assuming the IDM web application was deployed at `/openidm`.

```
/openidm/managed/type/id
```

The URI scheme for accessing a system object follows this convention:

```
/openidm/system/resource-name/type/id
```

An example of a system object in an LDAP directory might be:

```
/openidm/system/ldap/account/07b46858-56eb-457c-b935-cfe6ddf769c7
```

Important

For LDAP resources, you should *not* map the LDAP `dn` to the IDM `uidAttribute` (`_id`). The attribute that is used for the `_id` should be immutable. You should therefore map the LDAP `entryUUID` operational attribute to the IDM `_id`, as shown in the following excerpt of the provisioner configuration file:

```
...
"uidAttribute" : "entryUUID",
...
```

Object identifiers

Every managed and system object has an identifier (expressed as id in the URI scheme) that is used to address the object through the REST API. The REST API allows for client-generated and server-generated identifiers, through PUT and POST methods. The default server-generated identifier type is a UUID. If you create an object by using `POST`, a server-assigned ID is generated in the form of a UUID. If you create an object by using PUT, the client assigns the ID in whatever format you specify.

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

Content negotiation

The REST API fully supports negotiation of content representation through the `Accept` HTTP header. Currently, the supported content type is JSON. When you send a JSON payload, you must include the following header:

```
Accept: application/json
```

In a REST call (using the `curl` command, for example), you would include the following option to specify the noted header:

```
--header "Content-Type: application/json"
```

You can also specify the default UTF-8 character set as follows:

```
--header "Content-Type: application/json;charset=utf-8"
```

The `application/json` content type is not needed when the REST call does not send a JSON payload.

Conditional operations

The REST API 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 IDM's optimistic concurrency control system. Clients should make requests conditional in order to prevent inadvertent modification of the wrong version of an object.

REST API Conditional Operations

HTTP Header	Operation	Description
`If-Match: <rev>`	PUT	Update the object if the <rev> matches the revision level of the object.
`If-Match: *`	PUT	Update the object regardless of revision level.
`If-None-Match: <rev>`		Bad request.
`If-None-Match: *`	PUT	Create; fails if the object already exists.

HTTP Header	Operation	Description
When the conditional operations `If-Match`, `If-None-Match` are not used	PUT	Upsert; attempts a create, and then an update; if both attempts fail, return an error.

REST endpoints

Server configuration

IDM stores configuration objects in the repository, and exposes them under the context path `/openidm/config`. Single instance configuration objects are exposed under `/openidm/config/object-name`.

Multiple instance configuration objects are exposed under `/openidm/config/object-name/instance-name`. The following table outlines these configuration objects and how they can be accessed through the REST interface.

URI	HTTP Operation	Description
/openidm/config	GET	Returns a list of configuration objects.
/openidm/config/access	GET	Returns the current access configuration.
/openidm/config/audit	GET	Returns the current audit configuration.
/openidm/config/provisioner.openicf/provisioner-name	GET	Returns the configuration of the specified connector.
/openidm/config/selfservice/function	GET	Returns the configuration of the specified self-service feature, `registration`, `reset`, or `username`.
/openidm/config/router	PUT	Changes the router configuration. Modifications are provided with the `--data` option, in JSON format.
/openidm/config/object	PATCH	Changes one or more fields of the specified configuration object. Modifications are provided as a JSON array of patch operations.
/openidm/config/object	DELETE	Deletes the specified configuration object.
/openidm/config/object?_queryFilter=query	GET	Queries the specified configuration object. You cannot create custom predefined queries to query the configuration.

IDM supports REST operations to create, read, update, query, and delete configuration objects.

One entry is returned for each configuration object. To obtain additional information on the configuration object, include its `pid` or `_id` in the URL. The following example displays configuration information on the `sync` object, based on a deployment using the `sync-with-csv` sample:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/config/sync"
{
  "_id": "sync",
  "mappings": [
    {
      "name": "systemCsvfileAccounts_managedUser",
      "source": "system/csvfile/account",
      "target": "managed/user",
      "correlationQuery": {
        "type": "text/javascript",
        "source": "var query = {'_queryId' : 'for-userName', 'uid' : source.name};query;"
      },
      "properties": [
        {
          "source": "email",
          "target": "mail"
        },
        {
          "source": "firstname",
          "target": "givenName"
        },
        {
          "source": "lastname",
          "target": "sn"
        },
        {
          "source": "description",
          "target": "description"
        },
        {
          "source": "_id",
          "target": "_id"
        },
        {
          "source": "name",
          "target": "userName"
        },
        {
          "default": "Passw0rd",
          "target": "password"
        },
        {
          "source": "mobileTelephoneNumber",
          "target": "telephoneNumber"
        },
        {
          "source": "roles",
          "transform": {
            "type": "text/javascript",
```

```
            "source": "var _ = require('lib/lodash'); _.map(source.split(','), function(role)
            { return {'_ref': 'internal/role/' + role} });"
          },
          "target": "authzRoles"
        }
      ],
...
```

Managed users

User objects are stored in the repository and are exposed under the context path `/managed/user`. Many examples of REST calls related to this context path exist throughout this document. The following table lists available functionality associated with the `/managed/user` context path.

URI	HTTP Operation	Description
/openidm/managed/user?_queryFilter=true&_fields=_id	GET	Lists the IDs of all the managed users in the repository.
/openidm/managed/user?_queryFilter=true	GET	Lists all info for the managed users in the repository.
/openidm/managed/user?_queryFilter=filter	GET	Queries the managed user object with the defined filter.
/openidm/managed/user/_id	GET	Returns the JSON representation of a specific user.
/openidm/managed/user/_id	PUT	Creates a new user.
/openidm/managed/user/_id	PUT	Updates a user entry (replaces the entire entry).
/openidm/managed/user?_action=create	POST	Creates a new user.
/openidm/managed/user?_action=patch&_queryId=for-userName&uid=userName	POST	Updates a user (can be used to replace the value of one or more existing attributes).
/openidm/managed/user/_id	PATCH	Updates specified fields of a user entry.
/openidm/managed/user/_id	DELETE	Deletes a user entry.

For a number of sample commands that show how to manage users over REST, refer to Users.

Managed organizations

Organizations are exposed under the context path `/managed/organization`. The following table lists the REST commands associated with managed organizations.

URI	HTTP Operation	Description
/openidm/managed/organization?_queryFilter=true&_fields=_id	GET	Lists the IDs of all managed organizations.
/openidm/managed/organization?_queryFilter=filter	GET	Queries managed organizations with the defined filter.
/openidm/managed/organization/_id	GET	Returns the JSON representation of a specific organization.
/openidm/managed/organization/_id	PUT	Creates an organization with a user-defined ID.
/openidm/managed/organization/_id	PUT	Updates an organization (replaces the entire object).
/openidm/managed/organization?_action=create	POST	Creates a new organization with a system-generated ID.
/openidm/managed/organization/_id	DELETE	Deletes an organization.

For a number of sample commands that show how to manage organizations over REST, refer to Managed Organizations.

System objects

System objects, that is, objects that are stored in remote systems, are exposed under the `/openidm/system` context. IDM provides access to system objects over REST, as listed in the following table:

URI	HTTP Operation	Description
/openidm/system?_action=action-name	POST	`_action=availableConnectors` returns a list of the connectors that are available in `openidm/connectors` or in `openidm/bundle`. `_action=createCoreConfig` takes the supplied connector reference (`connectorRef`) and adds the configuration properties required for that connector. This generates a core connector configuration that you can use to create a full configuration with the `createFullConfig` action. `_action=createFullConfig` generates a complete connector configuration, using the configuration properties from the `createCoreConfig` action, and retrieving the object types and operation options from the resource, to complete the configuration. `_action=test` returns a list of all remote systems, with their status, and supported object types. `_action=testConfig` validates the connector configuration provided in the POST body. `_action=liveSync` triggers a liveSync operation on the specified source object. `_action=authenticate` authenticates to the specified system with the credentials provided.
/openidm/system/system-name?_action=action-name	POST	`_action=test` tests the status of the specified system.
/openidm/system/system-name/system-object?_action=action-name	POST	`_action=liveSync` triggers a liveSync operation on the specified system object. `_action=authenticate` authenticates to the specified system object, with the provided credentials. `_action=create` creates a new system object.
/openidm/system/system-name?_action=script&scriptId=script-name&scriptExecuteMode=resource	POST	`_action=script` runs the specified script on the system object.
/openidm/system/system-name/system-object?_queryId=query-all-ids	GET	Lists all IDs related to the specified system object, such as users, and groups.
/openidm/system/system-name/system-object?_queryFilter=filter	GET	Lists the item(s) associated with the query filter.
/openidm/system/system-name/system-object/id	PUT	Creates a system object, or updates the system object, if it exists (replaces the entire object).
/openidm/system/system-name/system-object/id	PATCH	Updates the specified fields of a system object.

URI	HTTP Operation	Description
/openidm/system/system-name/system-object/id	DELETE	Deletes a system object.

Note

When you create a system object with a PUT request (that is, specifying a client-assigned ID), you should specify the ID in the URL only and not in the JSON payload. If you specify a different ID in the URL and in the JSON payload, the request will fail, with an error similar to the following:

```
{
    "code":500,
    "reason":"Internal Server Error",
    "message":"The uid attribute is not single value attribute."
}
```

A `POST` request with a `patch` action is not currently supported on system objects. To patch a system object, you must send a `PATCH` request.

List available connector configurations:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=availableConnectors"
```

List remote systems, and their status:

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

Run liveSync on a specified system object:

Source Parameter

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=liveSync&source=system/ldap/account"
{
  "connectorData": {
    "nativeType": "integer",
    "syncToken": 0
  },
  "_rev": "00000000a92657c7",
  "_id": "SYSTEMLDAPACCOUNT"
}
```

Endpoint

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/ldap/account?_action=liveSync"
{
  "connectorData": {
    "nativeType": "integer",
    "syncToken": 0
  },
  "_rev": "00000000a92657c7",
  "_id": "SYSTEMLDAPACCOUNT"
}
```

Run a script on a system object:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/ldap/account?_action=script&_scriptId=addUser"
```

Authenticate to a system object

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "username" : "bjensen",
  "password" : "Passw0rd"
}' \
"http://localhost:8080/openidm/system/ldap/account?_action=authenticate"
{
  "_id": "fc252fd9-b982-3ed6-b42a-c76d2546312c"
}
```

Create a new system object

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--data '{
  "cn": "James Smith",
  "dn": "uid=jsmith,ou=people,dc=example,dc=com",
  "uid": "jsmith",
  "sn": "Smith",
  "givenName":"James",
  "mail": "jsmith@example.com",
  "description": "Created by IDM REST"}' \
--request POST \
"http://localhost:8080/openidm/system/ldap/account?_action=create"
{
  "telephoneNumber": null,
  "description": "Created by IDM REST",
  "mail": "jsmith@example.com",
  "givenName": "James",
  "cn": "James Smith",
  "dn": "uid=jsmith,ou=people,dc=example,dc=com",
  "uid": "jsmith",
  "ldapGroups": [],
  "sn": "Smith",
  "_id": "07b46858-56eb-457c-b935-cfe6ddf769c7"
}
```

Rename a system object

You can rename a system object simply by supplying a new naming attribute value in a PUT request. The PUT request replaces the entire object. The naming attribute depends on the external resource.

The following example renames an object on an LDAP server, by changing the DN of the LDAP object (effectively performing a modDN operation on that object). The example renames the user created in the previous example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--header "If-Match: *" \
--data '{
  "cn": "James Smith",
  "dn": "uid=jimmysmith,ou=people,dc=example,dc=com",
  "uid": "jimmysmith",
  "sn": "Smith",
  "givenName": "James",
  "mail": "jsmith@example.com"}' \
--request PUT \
"http://localhost:8080/openidm/system/ldap/account/07b46858-56eb-457c-b935-cfe6ddf769c7"
{
  "mail": "jsmith@example.com",
  "cn": "James Smith",
  "sn": "Smith",
  "dn": "uid=jimmysmith,ou=people,dc=example,dc=com",
  "ldapGroups": [],
  "telephoneNumber": null,
  "description": "Created by IDM REST",
  "givenName": "James",
  "uid": "jimmysmith",
  "_id": "07b46858-56eb-457c-b935-cfe6ddf769c7"
}
```

List IDs associated with a specific system object:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account?_queryId=query-all-ids"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 3,
  "result": [
    {
      "dn": "uid=jdoe,ou=People,dc=example,dc=com",
      "_id": "1ff2e78f-4c4c-300c-b8f7-c2ab160061e0"
    },
    {
      "dn": "uid=bjensen,ou=People,dc=example,dc=com",
      "_id": "fc252fd9-b982-3ed6-b42a-c76d2546312c"
    },
    {
      "dn": "uid=jimmysmith,ou=people,dc=example,dc=com",
      "_id": "07b46858-56eb-457c-b935-cfe6ddf769c7"
    }
  ]
}
```

Internal objects

You can manage the following internal objects over REST:

URI	HTTP Operation	Description
/openidm/internal/role?_queryFilter=true	GET	Lists all internal roles.
/openidm/internal/user?_queryFilter=true	GET	Lists internal users.
/openidm/internal/user/username	PUT	Adds a new internal user, or changes the password of an existing internal user.
/openidm/internal/user/username	PATCH	Adds or removes roles of an internal user.
/openidm/internal/role?_queryFilter=true&_fields=_id	GET	Lists internal roles.
/openidm/internal/role/role-id?_fields=*,authzMembers	GET	Lists internal and managed users with the specified internal role.

Schedules

Use the scheduler service to manage and monitor scheduled jobs.

You can access the scheduler service over REST, as indicated in the following table:

URI	HTTP Operation	Description
/openidm/scheduler?_action=validateQuartzCronExpression	POST	Validates a cron expression.
/openidm/scheduler/job/id	PUT	Creates or updates a schedule with the specified ID.
	GET	Obtains the details of the specified schedule.
	POST with ?_action=trigger *API V2 only*	Manually triggers the specified schedule.
	POST with ?_action=pause *API V2 only*	Suspends the specified schedule.
	POST with ?_action=resume *API V2 only*	Resumes the specified schedule.
	DELETE	Deletes the specified schedule.
/openidm/scheduler/job?_action=create	POST	Creates a schedule with a system-generated ID.
/openidm/scheduler/job?_queryFilter=query	GET	Queries the existing defined schedules.
/openidm/scheduler/job?_action=listCurrentlyExecutingJobs	POST	Returns a list of the jobs that are currently running.
/openidm/scheduler/job?_action=pauseJobs	POST	Suspends all scheduled jobs.
/openidm/scheduler/job?_action=resumeJobs	POST	Resumes all suspended scheduled jobs.
/openidm/scheduler/trigger?_queryFilter=query	GET	Queries the existing triggers.
/openidm/scheduler/trigger/id	GET	Obtains the details of the specified trigger.
/openidm/scheduler/acquiredTriggers	GET	Returns an array of the triggers that have been acquired, per node.
/openidm/scheduler/waitingTriggers	GET	Returns an array of the triggers that have not yet been acquired.

Scanning tasks

The task scanning mechanism lets you perform a batch scan for a specified date, on a scheduled interval, and then execute a task when this date is reached.

IDM provides REST access to the task scanner, as listed in the following table:

URI	HTTP Operation	Description
/openidm/taskscanner	GET	Lists all the scanning tasks, past and present.
/openidm/taskscanner/id	GET	Lists details of the given task.
/openidm/taskscanner?_action=execute&name=name	POST	Triggers the specified task scan run.
/openidm/taskscanner/id?_action=cancel	POST	Cancels the specified task scan run.

Audit logs

You can interact with the audit logs over REST, as shown in the following table. Queries on the audit endpoint must use `queryFilter` syntax.

URI	HTTP Operation	Description
/openidm/audit/recon?_queryFilter=true	GET	Displays the reconciliation audit log.
/openidm/audit/recon/id	GET	Reads a specific reconciliation audit log entry.
/openidm/audit/recon/id	PUT	Creates a reconciliation audit log entry.
/openidm/audit/recon?_queryFilter=/reconId+eq+"reconId"	GET	Queries the audit log for a particular reconciliation operation.
/openidm/audit/recon?_queryFilter=/reconId+eq+"reconId"+and+situation+eq+"situation"	GET	Queries the reconciliation audit log for a specific reconciliation situation.
/openidm/audit/sync?_queryFilter=true	GET	Displays the synchronization audit log.
/openidm/audit/sync/id	GET	Reads a specific synchronization audit log entry.
/openidm/audit/sync/id	PUT	Creates a synchronization audit log entry.
/openidm/audit/activity?_queryFilter=true	GET	Displays the activity log.

URI	HTTP Operation	Description
/openidm/audit/activity/id	GET	Returns activity information for a specific action.
/openidm/audit/activity/id	PUT	Creates an activity audit log entry.
/openidm/audit/activity?_queryFilter=transactionId=id	GET	Queries the activity log for all actions resulting from a specific transaction.
/openidm/audit/access?_queryFilter=true	GET	Displays the full list of auditable actions.
/openidm/audit/access/id	GET	Displays information on the specific audit item.
/openidm/audit/access/id	PUT	Creates an access audit log entry.
/openidm/audit/authentication?_queryFilter=true	GET	Displays a complete list of authentication attempts, successful and unsuccessful.
/openidm/audit/authentication?_queryFilter=/principal+eq+"principal"	GET	Displays the authentication attempts by a specified user.
/openidm/audit?_action=availableHandlers	POST	Returns a list of audit event handlers.
openidm/audit/config?_queryFilter=true	GET	Lists changes made to the configuration.

Reconciliation operations

You can interact with the reconciliation engine over REST, as shown in the following table:

URI	HTTP Operation	Description
/openidm/recon	GET	Lists all reconciliation runs, including those in progress. Inspect the `state` property to see the reconciliation status.
/openidm/recon?_action=recon&mapping=mapping-name	POST	Launches a reconciliation run with the specified mapping.
/openidm/recon?_action=reconById&mapping=mapping-name&id=id	POST	Restricts the reconciliation run to the specified ID.
/openidm/recon/id?_action=cancel	POST	Cancels the specified reconciliation run.

The following example runs a reconciliation for the mapping `systemHrdb_managedUser`:

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

Synchronization service

You can interact with the synchronization service over REST, as shown in the following table:

URI	HTTP Operation	Description
/openidm/sync?_action=getLinkedResources&resourceName=resource	POST	Provides a list of linked resources for the specified resource.
/openidm/sync/mappings?_queryFilter=true	GET	Returns a list of all configured mappings, in the order in which they will be processed.
/openidm/sync/queue?_queryFilter=filter	GET	Lists the queued synchronization events, based on the specified filter.
/openidm/sync/queue/eventID	DELETE	Deletes a queued synchronization event, based on its ID.

For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
"http://localhost:8080/openidm/sync?_action=getLinkedResources&resourceName=managed/user/
42f8a60e-2019-4110-a10d-7231c3578e2b"
[
  {
    "resourceName": "system/ldap/account/03496258-1c5e-40a0-8744-badc2500f262",
    "content": {
      "uid": "joe.smith1",
      "mail": "joe.smith@example.com",
      "sn": "Smith",
      "givenName": "Joe",
      "employeeType": [],
      "dn": "uid=joe.smith1,ou=People,dc=example,dc=com",
      "ldapGroups": [],
      "cn": "Joe Smith",
      "kbaInfo": [],
      "aliasList": [],
      "objectClass": [
        "top",
        "inetOrgPerson",
        "organizationalPerson",
        "person"
      ],
      "_id": "03496258-1c5e-40a0-8744-badc2500f262"
    },
    "linkQualifier": "default",
    "linkType": "systemLdapAccounts_managedUser"
  }
]
```

Scripts

You can interact with the script service over REST, as shown in the following table:

URI	HTTP Operation	Description
/openidm/script?_action=compile	POST	Compiles a script, to validate that it can be executed. Note that this action compiles a script, but does not execute it. A successful compilation returns `true`. An unsuccessful compilation returns the reason for the failure.
/openidm/script?_action=eval	POST	Executes a script and returns the result, if any.

The following example compiles, but does not execute, the script provided in the JSON payload:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "type": "text/javascript",
  "source": "source.mail ? source.mail.toLowerCase() : null"
}' \
"http://localhost:8080/openidm/script?_action=compile"
True
```

The following example executes the script referenced in the `file` parameter, with the provided input:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "type": "text/javascript",
  "file": "script/autoPurgeAuditRecon.js",
  "globals": {
    "input": {
      "mappings": ["%"],
      "purgeType": "purgeByNumOfRecordsToKeep",
      "numOfRecons": 1
    }
  }
}' \
"http://localhost:8080/openidm/script?_action=eval"
"Must choose to either purge by expired or number of recons to keep"
```

Privileges

Privileges are a part of internal roles, and can be created or modified using the REST calls specified in Internal objects. Additionally, `openidm/privilege` can be used for getting information about privileges on a resource as they apply to the authenticated user.

URI	HTTP Operation	Description
/openidm/privilege?_action=listPrivileges	POST	Lists an array of privilege paths for the authenticated user, with additional detail required by the admin UI.

URI	HTTP Operation	Description
/openidm/privilege/resource	GET	Lists the privileges for the logged in user associated with the given resource path.
/openidm/privilege/resource/guid	GET	Lists the privileges for the logged in user associated with the specified object.

Email

> **Note**
>
> To configure the email service, refer to Outbound email.

You can use the IDM outbound email service over REST at the `external/email` endpoint:

URI	HTTP Operation	Description
/openidm/external/email?_action=send	POST	Sends an email.
/openidm/external/email?_action=sendTemplate	POST	Sends an email template.

For complete examples, refer to Send mail using REST.

File upload

IDM supports a generic file upload service at the `file` endpoint. Files are uploaded either to the filesystem or to the repository. For information about configuring this service, and for command-line examples, refer to Upload files to the server.

IDM provides REST access to the file upload service, as listed in the following table:

URI	HTTP Operation	Description
/openidm/file/handler/	PUT	Uploads a file to the specified file handler. The file handler is either the repository or the filesystem and the context path is configured in the `conf/file-handler.json` file.
/openidm/file/handler/filename	GET	Returns the file content in a base 64-encoded string within the returned JSON object.
/openidm/file/handler/filename?_fields=content&_mimeType=mimeType	GET	Returns the file content with the specified MIME type.

URI	HTTP Operation	Description
/openidm/file/handler/filenamemimeType	DELETE	Deletes an uploaded file.

Bulk import

The bulk import service lets you import large numbers of entries from a CSV file into the IDM repository. You can import any managed object type, but you will generally use this service to import user entries. The following table shows the endpoints used by the bulk import service:

URI	HTTP Operation	Description
/openidm/csv/template?resourceCollection=managed/user	GET	Generates a CSV header row that you can use as a template for the import. You can safely remove generated columns for properties that are not required. Set the query parameters `_fields=header` and `_mimeType=text/csv` to download the header file.
/upload/csv/resourceCollection	POST	Uploads the file specified by the `--form` (`-F`) parameter to the specified resource collection. `?uniqueProperty=propertyName` is required. Generally, for `managed/user` objects, the `uniqueProperty` is `userName`. You can specify multiple comma-delimited values here to identify unique records; for example, `?uniqueProperty=firstName,lastName`. Example.
/openidm/csv/metadata/?_action=cleanupList	POST	Lists the import UUIDs that have error records or temporary records. These can be cleaned up to free up database space. If you clean up error records, you will no longer be able to download a CSV of failed import records.
/openidm/csv/metadata/importUUID?_action=cleanup	POST	Cleans up temporary import records for the specified import UUID. To also clean up error records, set the query parameter `?deleteErrorRecords=true`.
/openidm/csv/metadata/importUUID?_action=cancel	POST	Cancels the specified in-progress import.
/openidm/csv/metadata/importUUID	DELETE	Deletes the specified import record. This does not affect the data that was imported.

URI	HTTP Operation	Description
/openidm/csv/metadata?_queryFilter	GET	Queries bulk imports.
/openidm/csv/metadata/importUUID	GET	Reads the specified import record.
/export/csvImportFailures/importUUID	GET	Downloads a CSV file of failed import records. Returns 404 if there were no failures for the specified import UUID.

Server state

You can access information about the current state of the IDM instance through the `info` endpoint, as shown in the following table:

URI	HTTP Operation	Description
/openidm/info/features?_queryFilter=true	GET	Queries the available features in the server configuration.
/openidm/info/login	GET	Provides authentication and authorization details for the current user.
/openidm/info/ping	GET	Lists the current server state. Possible states are `STARTING`, `ACTIVE_READY`, `ACTIVE_NOT_READY`, and `STOPPING`.
/openidm/info/version	GET	Provides the software version of this IDM instance.

Social identity providers

URI	HTTP Operation	Description
/openidm/identityProviders	GET	Returns JSON details for all configured social identity providers.

URI	HTTP Operation	Description
/openidm/authentication	GET	Returns JSON details for all configured social identity providers, if the `SOCIAL_PROVIDERS` module is enabled. **Important** Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.
/openidm/managed/social_identity_provider	multiple	Supports access to social identity provider information.
/openidm/managed/user/social_identity_provider	GET	Supports a list of users associated with a specific social identity provider.
/openidm/managed/user/User_UUID/idps	multiple	Supports management of social identity providers by UUID.

Workflows

Workflow objects are exposed under the `/openidm/workflow` context path. IDM provides access to the workflow module over REST, as listed in the following table:

URI	HTTP Operation	Description
/openidm/workflow/execution?_action=message	POST	Invokes a message event. When the `processVariables` field is *non-null* the message is sent synchronously; otherwise, asynchronously.
/openidm/workflow/execution?_action=signal	POST	Invokes a signal event. When the `processVariables` field is *non-null* the message is sent synchronously; otherwise, asynchronously.
/openidm/workflow/execution?_action=trigger	POST	Triggers an execution asynchronously; for example, to continue a process instance that is waiting at a *Receive Task*. When the `transientVariables` field is *non-null* the trigger is sent synchronously; otherwise, asynchronously.
/openidm/workflow/execution?_queryFilter=true	GET	Queries the executions. Only supports `_queryFilter=true`.

URI	HTTP Operation	Description
/openidm/workflow/execution?_queryId=filtered-query&filter	GET	Returns a list of executions, based on the specified query filter. • `executionId` • `executionParentId` • `processDefinitionCategory` • `processDefinitionId` • `processDefinitionKey` • `processDefinitionName` • `processDefinitionVersion` • `processInstanceBusinessKey` • `processInstanceId` • `activityId` • `signalName` • `messageName` • `startedBefore` • `startedAfter` • `startedBy` • `processVariableName` • `processVariableValue` • `processVariableValueType` • `processVariableOperator` • `variableName` • `variableValue` • `variableValueType` • `variableOperator`
/openidm/workflow/job?_queryFilter=true	GET	Queries jobs. Only supports `_queryFilter=true`.
/openidm/workflow/job?_queryId=filtered-query&filter	GET	Returns a list of jobs, based on the specified query filter. You can't combine `timersOnly=true` with `messagesOnly=true` in the same query. • `jobId` • `processDefinitionId` • `processInstanceId` • `executionId` • `timersOnly` • `messagesOnly` • `withException` • `exceptionMessage` • `dueAfter` • `dueBefore`

URI	HTTP Operation	Description
/openidm/workflow/job/deadletter?_queryFilter=true	GET	Queries dead-letter jobs. Only supports `_queryFilter=true`.
/openidm/workflow/job/deadletter?_queryId=filtered-query&filter	GET	Returns a list of dead-letter jobs, based on the specified query filter. You can't combine `timersOnly=true` with `messagesOnly=true` in the same query. • `jobId` • `processDefinitionId` • `processInstanceId` • `executionId` • `timersOnly` • `messagesOnly` • `withException` • `exceptionMessage` • `dueAfter` • `dueBefore`
/openidm/workflow/job/deadletter/id?_action=execute	POST	Executes a dead-letter job. If successful, runs as a normal job.
/openidm/workflow/job/deadletter/id?_action=stacktrace	POST	Displays the stacktrace for a dead-letter job that triggered an exception.
/openidm/workflow/job/deadletter/id	DELETE	Deletes a dead-letter job.
/openidm/workflow/job/deadletter/id	GET	Reads a dead-letter job.
/openidm/workflow/job/id?_action=execute	POST	Forces the synchronous execution of a job, even if it is suspended.
/openidm/workflow/job/id?_action=stacktrace	POST	Displays the stacktrace for a job that triggered an exception.
/openidm/workflow/job/id	DELETE	Deletes a job.
/openidm/workflow/job/id	GET	Reads a job.
/openidm/workflow/model?_action=validate_bpmn	POST	Validates a BPMN 2.0 XML file.
/openidm/workflow/model	POST	Creates a new model. Omitting the `bpmnXML` field generates a template. Also refer to the `deploy` action.

URI	HTTP Operation	Description
/openidm/workflow/model?_queryFilter=query	GET	Queries the existing models. `bpmnXML` and `resourceMap` fields are omitted from results by default.
openidm/workflow/model/id?_action=deploy	POST	Deploys a model and creates associated process definitions. Existing process definition IDs will be returned if duplicate model detected. refer to `workflow/processdefinition` endpoints.
/openidm/workflow/model/id?_action=list_deployments	POST	Lists process definition IDs for model deployments.
/openidm/workflow/model/id	DELETE	Deletes a model.
/openidm/workflow/model/id	GET	Reads a model.
/openidm/workflow/model/id	PUT	Updates a model.
/openidm/workflow/processdefinition?_queryFilter=true	GET	Queries the process definitions. Only supports `_queryFilter=true`. Use the `READ` endpoint to get form-related fields.
/openidm/workflow/processdefinition?_queryId=filtered-query&filter	GET	Returns a list of workflow process definitions, based on the specified query filter. • `version` • `deploymentId` • `category` • `key` • `name` • `processDefinitionResourceName`
/openidm/workflow/processdefinition/id	DELETE	Deletes a process definition.
/openidm/workflow/processdefinition/id	GET	Reads a process definition with form-related fields included.
/openidm/workflow/processdefinition/procdefid/taskdefinition?_queryFilter=true	GET	Queries the task definitions. Only supports `_queryFilter=true`.
/openidm/workflow/processdefinition/procdefid/taskdefinition/id	GET	Reads a task definition.

URI	HTTP Operation	Description
/openidm/workflow/processinstance	POST	Creates a process instance. JSON request object must contain either the `_processDefinitionId` or `_key` fields, but not both. Additional JSON fields will be passed to the create-operation and utilized if applicable.
/openidm/workflow/processinstance?_queryFilter=true	GET	Queries the process instances. Only supports `_queryFilter=true`.
/openidm/workflow/processinstance?_queryId=filtered-query&filter	GET	Returns a list of workflow process instances, based on the specified query filter. • `processDefinitionId` • `processDefinitionKey` • `processInstanceBusinessKey` • `processInstanceId` • `superProcessInstanceId` • `finished` • `unfinished` • `involvedUserId` • `startUserId` • `startedAfter` • `startedBefore`
/openidm/workflow/processinstance/history?_queryFilter=true	GET	Queries the process instance history. Only supports `_queryFilter=true`.
/openidm/workflow/processinstance/history?_queryId=filtered-query&filter	GET	Returns a list of process instance history, based on the specified query filter. • `processDefinitionId` • `processDefinitionKey` • `processInstanceBusinessKey` • `processInstanceId` • `superProcessInstanceId` • `finished` • `unfinished` • `involvedUserId` • `startUserId` • `startedAfter` • `startedBefore`
/openidm/workflow/processinstance/history/id	DELETE	Deletes process instance history.

URI	HTTP Operation	Description
/openidm/workflow/processinstance/history/id	GET	Reads process instance history. `diagram` field returned when defined and requested in `_fields` parameter.
/openidm/workflow/processinstance/id?_action=migrate	POST	Migrates a process instance to a different process definition. To simulate the migration first, refer to the action `validateMigration`. `{ "processDefinitionId": "string", "variables": {}, "fromActivityIdMap": { "123": { "toActivityId": "string", "localVariables": {} } } }`
/openidm/workflow/processinstance/id?_action=validateMigration	POST	Simulates a process instance migration (`migrate` action).
/openidm/workflow/processinstance/id	DELETE	Deletes a process instance.
/openidm/workflow/processinstance/id	GET	Reads a process instance. `diagram` field returned when defined and requested in `_fields` parameter.
/openidm/workflow/taskinstance?_queryFilter=true	GET	Queries the task instances. Only supports `_queryFilter=true`.
/openidm/workflow/taskinstance?_queryId=filtered-query&filter	GET	Returns a list of task instances, based on the specified query filter. • `executionId` • `processDefinitionId` • `processDefinitionKey` • `processInstanceId` • `assignee` • `taskCandidateGroup` • `taskCandidateUser` • `name` • `owner` • `description` • `priority` • `unassigned`

URI	HTTP Operation	Description
/openidm/workflow/taskinstance?_queryId=unassignedTaskQuery	GET	Queries unassigned task instances for which the authenticated user is authorized to assign.
/openidm/workflow/taskinstance/history?_queryFilter=true	GET	Queries the task instance history. Only supports `_queryFilter=true`.
/openidm/workflow/taskinstance/history?_queryId=filtered-query&filter	GET	Returns a list of task instance history, based on the specified query filter. • `executionId` • `processDefinitionId` • `processDefinitionKey` • `processInstanceId` • `assignee` • `taskCandidateGroup` • `taskCandidateUser` • `taskId` • `taskName` • `owner` • `description` • `finished` • `unfinished` • `processFinished` • `processUnfinished` • `priority` • `deleteReason`
/openidm/workflow/taskinstance/history/id	GET	Reads a task instance history.
/openidm/workflow/taskinstance/id?_action=claim	POST	Assigns a task to a `userId`.
/openidm/workflow/taskinstance/id?_action=complete	POST	To complete a task, supply the required task parameters, as specified in the BPMN 2.0 XML file.
/openidm/workflow/taskinstance/id	DELETE	Deletes a task instance.
/openidm/workflow/taskinstance/id	GET	Reads a task instance.

URI	HTTP Operation	Description
/openidm/workflow/taskinstance/id	PUT	Updates a task instance. Must include one or more *supported* fields in JSON payload: • `assignee` • `description` • `name` • `owner` • `category` • `dueDate` • `priority`

The following examples list the defined workflows. For a workflow to appear in this list, the corresponding workflow definition must be in the `openidm/workflow` directory:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/workflow/processdefinition?_queryId=query-all-ids"
```

Depending on the defined workflows, the output will be something like the following:

```
{
  "result": [
    {
      "_id": "contractorOnboarding:1:5"
    },
    {
      "_id": "contractorOnboarding:2:9"
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

The following example invokes a workflow named "myWorkflow". The `foo` parameter is given the value `bar` in the workflow invocation:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "_key":"contractorOnboarding",
  "foo":"bar"
}' \
"http://localhost:8080/openidm/workflow/processinstance?_action=create"
```

Self-service reference

Reference documentation for the ForgeRock® Identity Management Self-Service REST API.

Note

This guide is reference documentation for IDM's self-contained service. If you are using the platform-based service using trees, refer to the Platform Self-Service documentation instead.
If you are just getting started, we recommend the platform-based version of self-service.

Quick Start

Self-Service Overview

Understand Self-Service Processes

Self-Registration

Configure User Self-Registration

Social Registration

Configure Registration Using Social Identity Providers

Progressive Profile

Progressive Profile Completion

Password Reset

Password Reset Process

Username Retrieval

Configure Username Retrieval

Additional Configuration

Additional configuration options for additional features such as reCAPTCHA, notifications, and the End User UI

Custom Stages

Add a Custom Stage to Self-Service

Stage Reference

Reference appendix of available self-service stages

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

This guide is intended for anyone developing a self-service application that acts as a client of ForgeRock Identity Management (IDM).

This guide is written with the expectation that you already have basic familiarity with the following topics:

- REST APIs
- JavaScript Object Notation (JSON) and basic IDM configuration

About user self-service

IDM provides a sample End User UI that implements a number of self-service processes, such as self-registration and password reset, based on a Self-Service REST API.

Self-service processes are configured in files named `selfservice-process-name.json` in your project's `conf` directory. Every self-service process steps through a series of *stages*, each with its own requirements, until the end of the process is reached or until the process exits with an exception. The flow through the stages differs, depending on how you have configured the process.

You can customize the default processes, or write your own custom processes by implementing the stages described in Self-service stage reference. For information about how self-service is implemented in the default End User UI, refer to Self-Service End User UI. For information on how to customize the End User UI, refer to the following Git repository: Identity Management (End User) - UI.

The Self-Service REST API supports only two HTTP requests:

- `GET` which obtains the requirements for that stage
- `POST` with `_action=submitRequirements`

The response to the `POST` request instructs the client how to proceed. The response can have one of two outcomes:

- Success—all requirements have been submitted and the process advances to the next stage.
- Failure—the behavior here differs by stage. Certain stages will exit with an exception, others will convert the exception into an error that the client must handle, others will simply return the requirements again.

The self-service process flow

Each self-service process advances through the stages in the order in which they are listed in the `stageConfigs` array in the process configuration file. The password reset process, for example, might include the following stages:

```
{
    "stageConfigs" : [
        {
            "name": "parameters",
            ...
        },
        {
            "name" : "userQuery",
            ...
        },
        {
            "name" : "validateActiveAccount",
            ...
        },
        {
            "name" : "emailValidation",
            ...
        },
        {
            "name" : "kbaSecurityAnswerVerificationStage",
            ...
        },
        {
            "name" : "resetStage",
            ..
        }
    ],
    ...
}
```

A process definition also includes an optional `snapshotToken` and `storage` parameter, for example:

```
{
    "stageConfigs" : [
        ...
    ],
    "snapshotToken" : {
        "type" : "jwt",
        "jweAlgorithm" : "RSAES_PKCS1_V1_5",
        "encryptionMethod" : "A128CBC_HS256",
        "jwsAlgorithm" : "HS256",
        "tokenExpiry" : 300
    },
    "storage" : "stateless"
}
```

The `snapshotToken` specifies the format of the token that is passed between the client and the server with each request. By default, this is a JWT token, stored statelessly, which means that the state is stored in the client, rather than on the server side. Because some legacy clients cannot handle the long URLs provided in a JWT token, you can store the snapshot token locally, as a `uuid` with the following configuration:

```
{
    ...
    "snapshotToken" : {
        "type" : "uuid"
    },
    "storage" : "local"
}
```

In this case, the 16-character token is stored in the IDM repository, in the `jsonstorage` table. To use this feature, copy `/path/to/openidm/samples/example-configurations/self-service/jsonstore.json` to your project's `conf/` directory. This file stores the configuration for the `uuid` token and includes the following settings:

- `entryExpireSeconds` —the amount of time before the password reset URL expires.
- `cleanupDwellSecondsliteral` —how often the server checks for and expires tokens.

  The value of `cleanupDwellSecondsliteral` should be a fraction of `entryExpireSeconds` so that expiration occurs close to the expected expiration time. The check is performed on a periodic basis.

For more information on the self-service tokens, refer to Tokens and User Self-Service.

If you do not include the `snapshotToken` and `storage` in the configuration, the default stateless configuration applies.

When a stage advances, it can optionally insert parameters into the process context or *state* for consumption by stages that occur later in the process. The snapshot token is essentially the state of the stage. It is the container in which `state`, `successAdditions` and other data are stored, and then returned to the client at the end of the process, as an encrypted blob named `token`.

Sample configurations for each default self-service process are available in the `/path/to/openidm/samples/example-configurations/self-service` directory.

Each self service process has a specific endpoint under `openidm/selfservice` with the name of the process; for example `openidm/selfservice/reset` for the Password Reset process. If you create a custom self-service process with a configuration file such as `selfservice-myprocess.json`, you produce an endpoint such as `{hostname}/openidm/selfservice/myprocess`.

All REST actions occur against that endpoint. For example, the following initial GET request against the password reset endpoint returns the requirements for the following stage:

```
curl \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-Password: anonymous" \
 --header "Accept-API-Version: resource=1.0" \
 --request GET \
 "http://localhost:8080/openidm/selfservice/reset"
{
  "_id": "1",
  "_rev": "-852427048",
  "type": "captcha",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Captcha stage",
    "type": "object",
    "required": [
      "response"
    ],
    "properties": {
      "response": {
        "recaptchaSiteKey": "6LcvE1IUAAAAAA5AI1SZzZJl-AlGvHM_dzUg-0_S",
        "description": "Captcha response",
        "type": "string"
      }
    }
  }
}
```

The default End User UI implements the following processes:

- Self-registration (under the endpoint `selfservice/registration`)
- Social registration (under the endpoint `selfservice/socialUserClaim`)

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

- Password reset (under the endpoint `selfservice/reset`)
- Forgotten username retrieval (under the endpoint `selfservice/username`)
- Progressive profile completion (under `selfservice/profile`)
- Security question updates (under `selfservice/kbaUpdate`)
- Terms and conditions (under `selfservice/termsAndConditions`)

The remainder of this guide describes each stage, its requirements, and expected responses.

Self-registration

This chapter describes the configuration, and the requests and responses for user self-registration.

Quick Start

Configuration

Configure User Self-Registration

Registration Form

Configure the User Self-Registration Form

Email Registration

Configure Emails for Self-Service Registration

User Preferences

Configure Synchronization Filters With User Preferences

Multiple Registration Flows

Configure Multiple User Self-Registration Flows

Examples

Example Self-Registration REST Requests

User self-registration

To set up basic user self-registration, you'll need at least the following configuration files:

ui-configuration.json

You can find this file in the default IDM project configuration directory, `openidm/conf`.

To enable self-service registration in the UI, enable the following boolean property in `ui-configuration.json`:

```
"selfRegistration" : true,
```

selfservice-registration.json

You can find a template version of this file in the following directory: `openidm/samples/example-configurations/self-service`. This includes the following properties:

- `allInOneRegistration`: determines whether IDM collects all user registration information in one or multiple pages. By default, it's set to true:

  ```
  "allInOneRegistration" : true,
  ```

- `stageConfigs`: configuration details for the stages included in the self-registration process. While the specific stages included may vary, most processes will include at least:
  - `idmUserDetails`: includes the IDM property for email addresses (`mail`), whether or not registration with social identity providers is enabled, and what data is required from new users, as described in User self-registration form.
  - `registrationPreferences`: lists preferences to include as defined in the `managed.json` file. For more information, refer to User preferences.
- `snapshotToken`: configuration details for the token used to store the user's details during the registration process.
- `storage`: determines how a user's details are stored for consumption by later stages in the registration process. By default, this is set to `stateless`.

Depending on how you configure User Self-Registration, you may need to set up additional configuration files, as discussed in User self-registration form.

Common components included in self-registration include:

- **Email validation**

  If you have included email verification, you must configure an outgoing email server. For details about the required addition to `selfservice-registration.json`, refer to Self-Service registration emails.

- **Security questions (KBA)**

  If you have configured security questions, users who self-register must create these questions during registration and answer them during the password reset process. You can also configure the system to force users who have been created during a reconciliation from an external data store to add security questions. The relevant code block is shown here, which includes security questions as a stage in the user self-registration process. For related configuration options, refer to Security questions.

```
{
    "name" : "kbaSecurityAnswerDefinitionStage",
    "kbaConfig" : null
},
```

- **Google ReCAPTCHA**

  If you've activated Google reCAPTCHA for user self-service registration, you'll refer to the following code block:

```
{
    "name" : "captcha",
    "recaptchaSiteKey" : "<siteKey>",
    "recaptchaSecretKey" : "<secretKey>",
    "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
},
```

  As suggested by the code, you'd substitute the actual `siteKey` and `secretKey` assigned by Google for your domain. For more information, refer to Google reCAPTCHA.

- **Terms & Conditions**

  If you've set up Terms & Conditions, users who self-register will have to accept them, based on criteria you create, as discussed in Terms & Conditions. If you've included Terms & Conditions with user self-registration, you'll refer to the following code block:

```
{
    "name" : "termsAndConditions"
},
```

  New users will have to manually accept these conditions before they complete the self-registration process.

- **Privacy & Consent**

  If you've configured Privacy & Consent, you'll refer to a code block with the `consent` name. The following code block includes template Privacy & Consent terms in English (`en`) and French (`fr`):

```
{
    "name" : "consent",
    "consentTranslations" : {
        "en" : "Please consent to sharing your data with whomever we like.",
        "fr" : "Veuillez accepter le partage de vos données avec les services de notre choix."
    }
},
```

> **Note**
>
> Substitute Privacy & Consent content that meets the requirements of your legal authorities.

For audit activity data related to user self-registration, refer to Query the Activity Audit Log

Configure self-registration using the admin UI

To configure user self-registration using the admin UI:

1. From the navigation bar, click **Configure > User Registration**.
2. On the **User Registration** page, enable **Enable User Registration**.

   > **Note**
   >
   > When you enable self-registration using the admin UI, IDM creates `selfservice-registration.json` if it doesn't already exist.

3. Configure options in the **Configure Registration Form** window:
   - **Identity Resource**, typically `managed/user`.
   - **Identity Email Field**, typically `mail` or `email`.
   - **Success URL** for the End User UI. Users who successfully log in are redirected to this URL. By default, `{hostname}/#dashboard/`.
   - **Preferences**, which set up default marketing preferences for new users. New users can change these preferences during registration, or from the End User UI.
   - **Advanced Options > Snapshot Token**, typically JSON Web Token (**JWT**).
   - **Advanced Options > Token Lifetime (seconds)**, with a default of **300** seconds.
4. Click **Save**.

   Now that User Registration is active, three tabs display on the **User Registration** page:
   - **Registration Form**, as described in User self-registration form.
   - **Social**, as described in Social registration.
   - **Options**, as described in Additional configuration.

Managing user self-registration over REST

To display the current user self-registration configuration over REST, run the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/config/selfservice/registration"
```

Unless you have disabled file writes, the output matches the contents of your project's `selfservice-registration.json` file.

To update the configuration over REST, include the desired file contents:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PUT \
--data '{ <Insert file contents here> }' \
"http://localhost:8080/openidm/config/selfservice/registration"
```

User self-registration form

During user self-registration, IDM displays user attributes on the user registration form, as defined in the `selfservice-registration.json` file. You can modify the displayed properties in the `registrationProperties` code block:

```
"registrationProperties" : [
    "userName",
    "givenName",
    "sn",
    "mail"
],
```

To add user attributes to the user self-registration form using the admin UI:

1. From the navigation bar, click **Configure > User Registration**.
2. On the **User Registration** page, select the **Registration Form** tab.
3. Below the list of attributes, click the drop-down list, select an attribute, and click **Add**.

> **Note**
>
> This action also adds the attribute to the `registrationProperties` code block.

The user self-registration form displays attributes in the listed order.

You can also set up user self-registration via configuration files, as described in the following table:

User Self-Registration Configuration Files

File Name	Description
`external.email.json`	To enable email verification, you must configure an outgoing email server.
`managed.json`	You can customize user self-registration based on entries in this file. To change the labels seen by end users, change the associated `title`.
`policy.json`	For more information, refer to Custom policies for self-registration and password reset.

File Name	Description
`selfservice.kba.json`	refer to Security questions.
`selfservice-registration.json`	refer to User self-registration.
`ui-configuration.json`	refer to User self-registration.

Self-Service registration emails

To configure self-service registration emails, add the following code block to the `selfservice-registration.json` file:

```
{
    "name" : "emailValidation",
    "identityEmailField" : "mail",
    "emailServiceUrl" : "external/email",
    "emailServiceParameters" : {
        "waitForCompletion" : false
    },
    "from" : "info@example.com",
    "subject" : "Register new account",
    "mimeType" : "text/html",
    "subjectTranslations" : {
        "en" : "Register new account",
        "fr" : "Créer un nouveau compte"
    },
    "messageTranslations" : {
        "en" : "<h3>This is your registration email.</h3><h4><a href=\"%link%\">Email verification link</a></h4>",
        "fr" : "<h3>Ceci est votre mail d'inscription.</h3><h4><a href=\"%link%\">Lien de vérification email</a></
h4>"
    },
    "verificationLinkToken" : "%link%",
    "verificationLink" : "https://localhost:8443/#/registration/"
},
```

The code block includes default registration email messages in English (`en`) and French (`fr`). The `verificationLink` sent with the email takes users to the IDM self-registration URL.

As noted in Managing User Self-Registration Over REST, you can make these changes over the endpoint URI: `/openidm/config/selfservice/registration`

To configure self-service registration emails using the admin UI:

1. From the navigation bar, click **Configure > User Registration**.
2. On the **User Registration** page, select the **Options** tab.
3. Enable **Email Validation**.

> **Note**
>
> If the **Email Validation** option is disabled, outbound email has not been configured. Click **Configure Here**, and refer to Outbound email for more information.

4. In the **Configure Validation Email** window, enter the necessary information, and click **Save**.

   Self-registration email changes made using the admin UI are saved to the `selfservice-registration.json` file.

User preferences

You can set up preferences for managed users, such as those related to marketing and news updates. You can then use those preferences as a filter when reconciling users to a target repository.

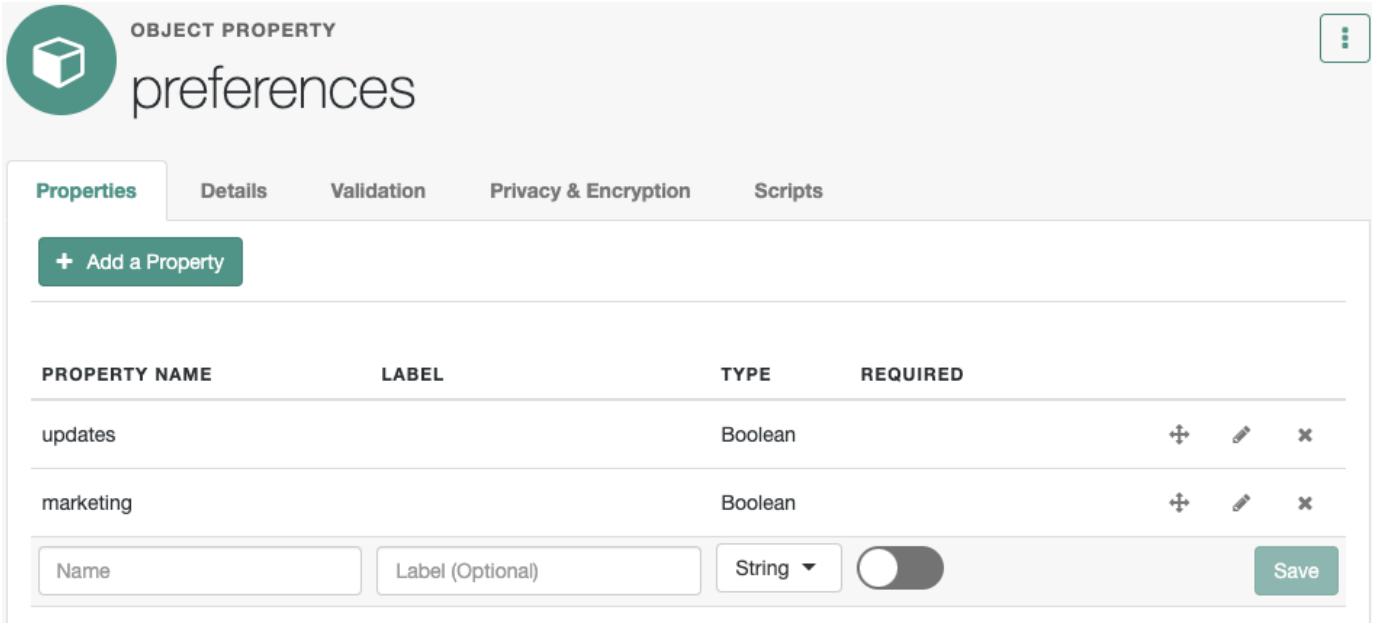
In the default project, common marketing preference options are included for the managed user object. You can see these preferences in the `managed.json` file:

```
"preferences" : {
    "title" : "Preferences",
    "description" : "Preferences",
    "viewable" : true,
    "searchable" : false,
    "userEditable" : true,
    "type" : "object",
    "usageDescription" : "",
    "isPersonal" : false,
    "properties" : {
        "updates" : {
            "description" : "Send me news and updates",
            "type" : "boolean"
        },
        "marketing": {
            "description" : "Send me special offers and services",
            "type" : "boolean"
        }
    },
    "order": [
        "updates",
        "marketing"
    ],
    "required": []
},
```

To view these preferences using the admin UI:

1. From the navigation bar, click **Configure > Managed Objects**.
2. On the **Managed Objects** page, click **User**.
3. Select the **Properties** tab, scroll down the properties list, and click **preferences**.

   The **Managed Objects > user > preferences** properties display:



Review preferences as an end user

When regular users log in to the End User UI, they'll refer to the preferences described in User preferences. When they accept the preferences, their managed user objects are updated with entries similar to the following:

```
"preferences" : {
    "updates" : true,
    "marketing" : true
},
```

User preferences and reconciliation

You can configure user preferences as a filter for reconciliation. For example, if some users don't want marketing emails, you can filter those users out of any reconciliation operation.

To configure user preferences as a reconciliation filter using the admin UI:

1. From the navigation bar, click **Configure > Mappings**, and select a mapping.
2. Click the **Association** tab, and expand the **Individual Record Validation** node.
3. From the **Valid Source** drop-down list, select **Validate based on user preferences**.
4. Select the applicable preferences checkboxes.

   For example, if you select the **Send me news and updates** checkbox, users who have opted-in to that preference will be reconciled from the source to the target repository.
5. Click **Save**.

> **Note**
>
> What IDM does during this reconciliation depends on the policy associated with the `UNQUALIFIED` situation for a `validSource`. The default action is to delete the target object (user). For more information, refer to How IDM assesses synchronization situations.

Alternatively, edit the mapping file directly. The following excerpt of a mapping file includes `preferences` as conditions to define a `validSource` on an individual record validation. IDM applies these conditions at the next reconciliation.

```
"validSource" : {
    "type" : "text/javascript",
    "globals" : {
        "preferences" : [
            "updates",
            "marketing"
        ]
    },
    "file" : "ui/preferenceCheck.js"
},
"validTarget" : {
    "type" : "text/javascript",
    "globals" : { },
    "source" : ""
}
```

Multiple user self-registration flows

You can set up multiple self-registration flows, with features limited only by the capabilities listed in Self-registration.

> **Note**
>
> Multiple self-registration flows, and customization of the End User UI beyond what is described in this document (and the noted public Git repository), are *not* supported.
> For additional information on customizing the End User UI, refer to the following ForgeRock Git repository: ForgeRock/end-user-ui: Identity Management (End User).

For example, you may want to set up different portals for regular employees and contractors. You'd configure each portal with different self-registration flows, managed by the same IDM backend. Each portal would use the appropriate registration API.

To prepare for this section, you'll need a `selfservice-registration.json` file. You can find a copy in the following directory: `/path/to/openidm/samples/example-configurations/self-service`.

To avoid errors when using this file, you should either:

- Copy the following files from the same directory:

  `selfservice.terms.json`
  `selfservice-termsAndConditions.json`

- Delete the `termsAndConditions` code block from the respective `selfservice-registration*.json` files.

User self-registration is normally coded in the `selfservice-registration.json` file. In preparation, copy this file to the `selfservice-registration*.json` to the names shown in the following list:

- Employee Portal
  - Configuration file: `selfservice-registrationEmployee.json`
  - URL: `https://localhost:8443/openidm/selfservice/registrationEmployee`
  - `verificationLink` : `https://localhost:8443/#/registrationEmployee`
- Contractor Portal
  - Configuration file: `selfservice-registrationContractor.json`
  - URL: `https://localhost:8443/openidm/selfservice/registrationContractor`
  - `verificationLink` : `https://localhost:8443/#/registrationContractor`

Edit the configuration file for each portal.

1. Modify the `verificationLink` URL associated with each portal as described.
2. Edit your access configuration (`conf/access.json`), by adding an endpoint for each new self-service registration file, after the `selfservice/registration` section. For example, the following code excerpt would apply to the `registrationEmployee` and `registrationContractor` endpoints:

```
{
    "pattern"    : "selfservice/registrationEmployee",
    "roles"      : "*",
    "methods"    : "read,action",
    "actions"    : "submitRequirements"
},
{
    "pattern"    : "selfservice/registrationContractor",
    "roles"      : "*",
    "methods"    : "read,action",
    "actions"    : "submitRequirements"
},
```

3. Modify the functionality of each selfservice-registration*.json file as desired. For guidance, refer to the sections noted in the following table:

Configuring `selfservice-registration*.json` *Files for Different Portals*

Feature	Code Block	Link
Social Registration	`"socialRegistrationEnabled" : true,`	Social registration

Feature	Code Block	Link

Properties requested during self-registration	<pre>"registrationProperties" : [
	    "userName",
	    "givenName",
	    "sn",
	    "mail"
	],</pre>
		User self-registration form

Terms & Conditions	<pre>{
	    "name" : "termsAndConditions"
	}</pre>
		Terms & Conditions

Privacy & Consent	<pre>{
	    "name" : "consent",
	    "consentTranslations" : {
	        "en" : "substitute appropriate Privacy & Consent wording",
	        "fr" : "substitute appropriate Privacy & Consent wording, in French"
	    }
	},</pre>
		Privacy and consent

reCAPTCHA	<pre>{
	    "name" : "captcha",
	    "recaptchaSiteKey" : "<siteKey>",
	    "recaptchaSecretKey" : "<secretKey>",
	    "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
	}</pre>
		Google reCAPTCHA

Email Validation		Self-Service registration emails

Security Questions	<pre>{
	    "name" : "kbaSecurityAnswerDefinitionStage",
	    "kbaConfig" : null
	},</pre>
		Security questions

If you leave out the code blocks associated with the feature, you won't refer to that feature in the self-service registration flow. In that way, you can set up different self-service registration flows for the Employee and Contractor portals.

Once you've configured both portals, you can make REST calls to both URLs: `https://localhost:8443/openidm/selfservice/registrationEmployee`
`https://localhost:8443/openidm/selfservice/registrationContractor`

For more advice on how you can create custom registration flows, refer to the following public ForgeRock Git repository: Identity Management (End User) - UI.

> **Note**
>
> The changes described in this section require changes to the End User UI source code as described in the noted public Git repository. Pay particular attention to the instructions associated with the `Registration.vue` file.

Self-registration REST requests

The REST calls shown in this chapter assume that user registration is enabled with the default security questions, and that the configuration is similar to that shown in the sample registration configuration file (`samples/example-configurations/self-service/selfservice-registration.json`):

```
{
    "allInOneRegistration" : true,
    "stageConfigs" : [
        {
            "name": "parameters",
            "parameterNames" : [
                "returnParams"
            ]
        },
        {
            "name" : "idmUserDetails",
            "identityEmailField" : "mail",
            "socialRegistrationEnabled" : true,
            "identityServiceUrl" : "managed/user",
            "registrationProperties" : [
                "userName",
                "givenName",
                "sn",
                "mail"
            ],
            "registrationPreferences": ["marketing", "updates"]
        },
        {
            "name" : "termsAndConditions"
        },
        {
            "name" : "emailValidation",
            "identityEmailField" : "mail",
            "emailServiceUrl" : "external/email",
            "emailServiceParameters" : {
                "waitForCompletion" : false
            },
            "from" : "info@admin.org",
            "subject" : "Register new account",
            "mimeType" : "text/html",
            "subjectTranslations" : {
                "en" : "Register new account",
                "fr" : "Créer un nouveau compte"
            },
            "messageTranslations" : {
                "en" : "<h3>This is your registration email.</h3><h4><a href=\"%link%\">Email verification link</a></
h4>",
                "fr" : "<h3>Ceci est votre email d'inscription.</h3><h4><a href=\"%link%\">Lien de vérification
email</a></h4>"
            },
            "verificationLinkToken" : "%link%",
            "verificationLink" : "https://idm.example.com:8443/#/registration/"
        },
        {
            "name" : "kbaSecurityAnswerDefinitionStage",
            "kbaConfig" : null
        },
        {
            "name" : "selfRegistration",
            "identityServiceUrl" : "managed/user"
        },
        {
            "name" : "localAutoLogin",
            "successUrl" : "",
            "identityUsernameField": "userName",
```

```
            "identityPasswordField": "password"
        }
    ],
    "storage" : "stateless"
}
```

1. The client loads the initial registration form. The server returns the `initial` tag to indicate the start of the registration process:

```
curl \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-Password: anonymous" \
 --header "X-OpenIDM-NoSession: true" \
 --request GET \
 "https://idm.example.com:8443/openidm/selfservice/registration"
{
  "_id": "1",
  "_rev": "1113597344",
  "type": "parameters",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Parameters",
    "type": "object",
    "properties": {
      "returnParams": {
        "description": "Parameter named 'returnParams'",
        "type": "string"
      }
    }
  }
}
```

   The client sends an empty POST request with the `submitRequirements` action.

   The server returns the following:

   - The `initial` tag to indicate the start of the registration process.
   - A `token` that must be provided in subsequent steps.
   - A JSON `requirements` object that must be provided in subsequent steps.

```
curl \
 --header "Content-type: application/json" \
 --header "X-OpenIDM-Password: anonymous" \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-NoSession: true" \
 --request POST \
 --data '{"input":{"input":{}}}' \
 https://idm.example.com:8443/openidm/selfservice/registration?_action=submitRequirements
{
  "type":"allInOneRegistration",
  "tag":"initial",
  "requirements":{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "description":"All-In-One Registration",
    "type":"object",
    "properties":{
      "response":{
        "recaptchaSiteKey":"6Lf…1ry",
        "description":"Captcha response",
        "type":"string"
      },
      "kba":{
        "type":"array",
        "minItems":2,
        "items":{
          "type":"object",
          "oneOf":[
            {
              "$ref":"#/definitions/systemQuestion"
            },
            {
              "$ref":"\#/definitions/userQuestion"
            }
          ]
        },
        "questions":[
          {
            "question":{
              "en":"What’s your favorite color?",
              "en_GB":"What is your favourite colour?",
              "fr":"Quelle est votre couleur préférée?"
            },
            "id":"1"
          },
          {
            "question":{
              "en":"Who was your first employer?"
            },
            "id":"2"
          }
        ]
      },
      "user":{
        "default":{
```

```
        },
        "description":"User Object",
        "type":"object"
      },
      "accept":{
        "description":"Accept",
        "type":"string"
      }
    },
    "required":[
      "response",
      "accept",
      "kba"
    ],
    "terms":"These are our terms and conditions",
    "termsVersion":"1.0",
    "uiConfig":{
      "displayName":"We have updated our terms",
      "purpose":"To proceed, accept these terms",
      "buttonText":"Accept"
    },
    "createDate":"2018-11-05T13:14:00.540Z",
    "definitions":{
      "systemQuestion":{
        "description":"System Question",
        "type":"object",
        "required":[
          "questionId",
          "answer"
        ],
        "properties":{
          "questionId":{
            "description":"Id of predefined question",
            "type":"string"
          },
          "answer":{
            "description":"Answer to the referenced question",
            "type":"string"
          }
        },
        "additionalProperties":false
      },
      "userQuestion":{
        "description":"User Question",
        "type":"object",
        "required":[
          "customQuestion",
          "answer"
        ],
        "properties":{
          "answer":{
             "description":"Answer to the question",
             "type":"string"
          },
          "customQuestion":{
```

```
            "description":"Question defined by the user",
            "type":"string"
          }
        },
        "additionalProperties":false
      },
      "providers":{
        "type":"array",
        "items":{
          "type":"object",
          "oneOf":[
          ]
        }
      }
    },
    "socialRegistrationEnabled":false,
    "registrationForm":null,
    "registrationProperties":{
      "properties":{
        "userName":{
          "title":"Username",
          "description":"Username",
          "viewable":true,
          "type":"string",
          "searchable":true,
          "userEditable":true,
          "usageDescription":"",
          "isPersonal":true,
          "policies":[
            {
              "policyId" : "minimum-length",
              "params" : {
                "minLength" : 1
              }
            },
            {
              "policyId":"unique"
            },
            {
              "policyId":"no-internal-user-conflict"
            },
            {
              "policyId":"cannot-contain-characters",
              "params":{
                "forbiddenChars":[
                  "/"
                ]
              }
            }
          ]
        },
        "givenName":{
          "title":"First Name",
          "description":"First Name",
          "viewable":true,
```

```
          "type":"string",
          "searchable":true,
          "userEditable":true,
          "usageDescription":"",
          "isPersonal":true
        },
        "sn":{
          "title":"Last Name",
          "description":"Last Name",
          "viewable":true,
          "type":"string",
          "searchable":true,
          "userEditable":true,
          "usageDescription":"",
          "isPersonal":true
        },
        "mail":{
          "title":"Email Address",
          "description":"Email Address",
          "viewable":true,
          "type":"string",
          "searchable":true,
          "userEditable":true,
          "usageDescription":"",
          "isPersonal":true,
          "policies":[
            {
               "policyId":"valid-email-address-format"
            }
          ]
        }
      },
      "required":[
        "userName",
        "givenName",
        "sn",
        "mail"
      ]
    },
    "registrationPreferences":{
      "updates":{
        "description":"Send me news and updates",
        "type":"boolean"
      },
      "marketing":{
        "description":"Send me special offers and services",
        "type":"boolean"
      }
    },
    "stages":[
      "captcha",
      "termsAndConditions",
      "kbaSecurityAnswerDefinitionStage",
```

```
      "idmUserDetails"
    ]
  },
  "token":"eyJ0eXAiOiJKV1QiLCJjdHkiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ZXlKMGVYQ…2h-k"
}
```

2. The client sends a POST request with the requirements. The server responds with a request for the emailed code:

```
curl \
 --header "Content-type: application/json" \
 --header "X-OpenIDM-Password: anonymous" \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-NoSession: true" \
 --request POST \
 --data '{
  "input":{
    "user":{
      "userName":"bjensen",
      "givenName":"Babs",
      "sn":"Jensen",
      "mail":"babs.k.jensen@gmail.com",
      "preferences":{
        "updates":false,
        "marketing":false
      },
      "password":"Passw0rd"
    },
    "kba":[
      {
        "answer":"red",
        "questionId":"1"
      },
      {
        "answer":"forgerock",
        "questionId":"2"
      }
    ],
    "response":"03AMGVjXggloUomtJx2Q0_wAjzyb9lN3LJBRIN67O85eGJIejO6WMlZGZ2jqnz…",
    "g-recaptcha-response":"03AMGVjXggloUomtJx2Q0_wAjzyb9lN3LJBRIN67O85eGJIejO…",
    "accept":"true"
  },
  "token":"eyJ0eXAiOiJKV1QiLCJjdHkiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ZXlKMGVYQWlPa…"
}' \
https://idm.example.com:8443/openidm/selfservice/registration?_action=submitRequirements
{
  "type":"emailValidation",
  "tag":"validateCode",
  "requirements":{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "description":"Verify emailed code",
    "type":"object",
    "required":[
      "code"
    ],
    "properties":{
      "code":{
        "description":"Enter code emailed",
        "type":"string"
```

```
      }
    }
  },
  "token":"eyJ0eXAiOiJKV1QiLCJjdHkiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ZXlKMGVYQWl…"
}
```

Note

By default, the snapshot token expires after 300 seconds. If the delay between the first request and the second request is greater than that period, the snapshot token will be invalid and the initial request must be sent again to obtain a fresh snapshot token. You can change the snapshot token expiration time in the self-service process configuration file (`selfservice-registration.json` in this case).
The following excerpt of the configuration file shows the default `snapshotToken` configuration. To change the expiration time, set the `tokenExpiry` property:

```
"snapshotToken" : {
    "type" : "jwt",
    "jweAlgorithm" : "RSAES_PKCS1_V1_5",
    "encryptionMethod" : "A128CBC_HS256",
    "jwsAlgorithm" : "HS256",
    "tokenExpiry" : 300
},
```

3. The email verification link redirects to:

   `https://idm.example.com:8443/#/registration/&token=eyJ0e..."`

   The client is registered and logged into the End User UI.

Social registration

Important

Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

IDM provides a standards-based solution for social authentication requirements, based on the OAuth 2.0 and OpenID Connect 1.0 standards. They are similar, as OpenID Connect 1.0 is an authentication layer built on OAuth 2.0.

This chapter describes how to configure IDM to register and authenticate users with multiple social identity providers.

To configure different social identity providers, you'll take the same general steps:

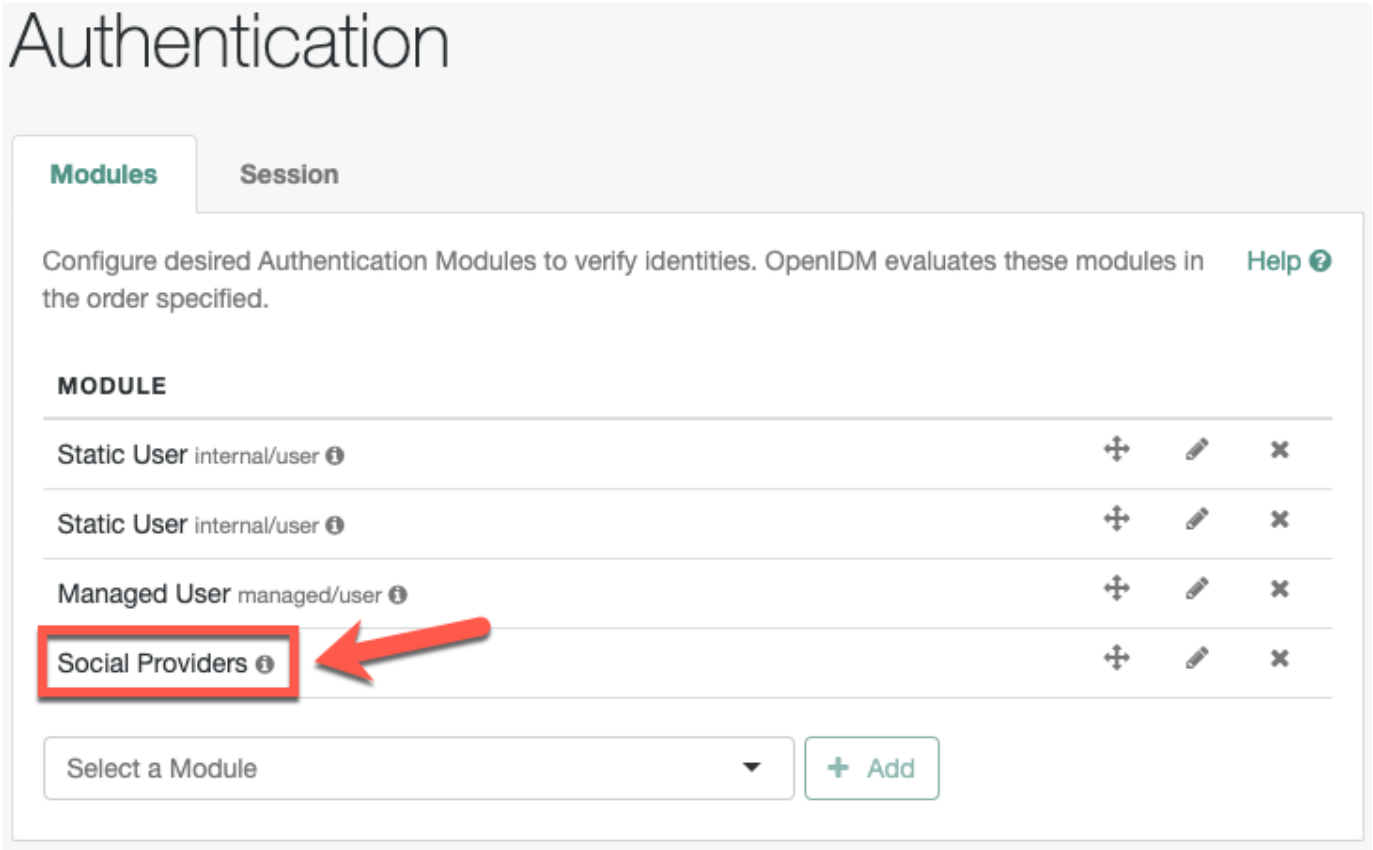
1. Enable the social providers authentication module.
2. Set up the provider. You'll need information such as a `Client ID` and `Client Secret` to set up an interface with IDM.
3. Configure the provider on IDM.
4. Set up User Registration. Activate `Social Registration` in the applicable admin UI screen or configuration file.
5. After configuration is complete, test the result. For a common basic procedure, refer to Test social identity providers.

Enable the social providers authentication module

You must enable the social providers authentication module before using social registration:

1. From the navigation bar, click **Configure > Authentication**.
2. On the **Authentication** page, click the **Modules** tab.
3. From the **Select a module** drop-down list, select **Social Providers**, and click **Add**.
4. In the **New Social Providers Authentication Module** window, make sure **Module Enabled** is enabled.
5. Make changes as necessary, and click **Save**.

   Social Providers now displays in the **Module** list.



6. Copy `/path/to/openidm/samples/example-configurations/self-service/identityProviders.json` to your project's `conf/` directory.

To understand how data is transmitted between IDM and a social identity provider, read OpenID connect authorization code flow.

> **Note**
>
> For all social identity providers, set up a FQDN for IDM, along with information in a DNS server, or system `hosts` files. For test purposes, FQDNs that comply with RFC 2606, such as `localhost` and `openidm.example.com`, are acceptable.

When you've configured one or more social identity providers, you can activate the **Social Registration** option in **User Registration**. This action adds:

- The following setting to the `selfservice-registration.json` configuration file:

```
"socialRegistrationEnabled" : true,
```

- The `selfservice-socialUserClaim.json` configuration file, discussed in Account Claiming.

Under the **Social** tab, you'll refer to a list of property mappings as defined in the `selfservice.propertymap.json` file.

One or more `source` properties in this file takes information from a social identity provider. When a user registers with their social identity account, that information is reconciled to the matching `target` property for IDM. For example, the `email` property from a social identity provider is normally reconciled to the IDM managed user `mail` property.

OpenID connect authorization code flow

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

The OpenID Connect Authorization Code Flow specifies how IDM (Relying Party) interacts with the OpenID Provider (Social ID Provider), based on the use of the OAuth 2.0 authorization grant. The following sequence diagram illustrates successful processing from the authorization request, through grant of the authorization code, access token, ID token, and provisioning from the social identity provider to IDM.

IDM supports OpenID Connect for Social Identity Providers

Figure 1. IDM supports OpenID Connect for Social Identity Providers

The following list describes details of each item in the authorization flow:

1. A user navigates to the IDM End User UI, and selects the `Sign In` link for the desired social identity provider.
2. IDM prepares an authorization request.
3. IDM sends the request to the Authorization Endpoint that you configured for the social identity provider, with a Client ID.
4. The social identity provider requests end user authentication and consent.
5. The end user transmits authentication and consent.
6. The social identity provider sends a redirect message, with an authorization code, to the end user's browser. The redirect message goes to an `oauthReturn` endpoint, configured in `ui.context-oauth.json` in your project's `conf/` directory.

   When you configure a social identity provider, you'll find the endpoint in the applicable configuration file with the following property: `redirectUri`.
7. The browser transmits the redirect message, with the authorization code, to IDM.
8. IDM records the authorization code, and sends it to the social identity provider Token Endpoint.

9. The social identity provider token endpoint returns access and ID tokens.
10. IDM validates the token, and sends it to the social identity provider User Info Endpoint.
11. The social identity provider responds with information on the user's account, that IDM can provision as a new Managed User.

You'll configure these credentials and endpoints, in some form, for each social identity provider.

Many social identity providers, one schema

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

Most social identity providers include common properties, such as name, email address, icon configuration, and location.

IDM includes two sets of property maps that translate information from a social identity provider to your managed user objects. These property maps are as follows:

- The `identityProviders.json` file includes a `propertyMap` code block for each supported provider. This file maps properties from the provider to a generic managed user object. You should not customize this file. To use this file, copy `/path/to/openidm/samples/example-configurations/self-service/identityProviders.json` to your project's `conf/` directory.
- The `selfservice.propertymap.json` file translates the generic managed user properties to the managed user schema that you have defined in `managed.json` . If you have customized the managed user schema, this is the file that you must change, to indicate how your custom schema maps to the generic managed user schema.

Examine `conf/identityProviders.json` . The following excerpt shows the Facebook `propertyMap` :

```
"propertyMap" : [
   {
      "source" : "id",
      "target" : "id"
   },
   {
      "source" : "name",
      "target" : "displayName"
   },
   {
      "source" : "first_name",
      "target" : "givenName"
   },
   {
      "source" : "last_name",
      "target" : "familyName"
   },
   {
      "source" : "email",
      "target" : "email"
   },
   {
      "source" : "email",
      "target" : "username"
   },
   {
      "source" : "locale",
      "target" : "locale"
   }
]
```

The source lists the Facebook property, the target lists the corresponding property for a generic managed user.

IDM then processes that information through the `selfservice.propertymap.json` file, where the source corresponds to the generic managed user and the target corresponds to your customized managed user schema (defined in your project's `managed.json` file).

```
{
   "properties" : [
      {
         "source" : "givenName",
         "target" : "givenName"
      },
      {
         "source" : "familyName",
         "target" : "sn"
      },
      {
         "source" : "email",
         "target" : "mail"
      },
      {
         "source" : "postalAddress",
         "target" : "postalAddress",
         "condition" : "/object/postalAddress  pr"
      },
      {
         "source" : "addressLocality",
         "target" : "city",
         "condition" : "/object/addressLocality  pr"
      },
      {
         "source" : "addressRegion",
         "target" : "stateProvince",
         "condition" : "/object/addressRegion  pr"
      },
      {
         "source" : "postalCode",
         "target" : "postalCode",
         "condition" : "/object/postalCode  pr"
      },
      {
         "source" : "country",
         "target" : "country",
         "condition" : "/object/country  pr"
      },
      {
         "source" : "phone",
         "target" : "telephoneNumber",
         "condition" : "/object/phone  pr"
      },
      {
         "source" : "username",
         "target" : "userName"
      }
   ]
}
```

Tip

To take additional information from a social identity provider, make sure the property is mapped through the `identityProviders.json` and `selfservice.propertymap.json` files.

Several of the property mappings include a `pr` presence expression which is a filter that returns all records with the given attribute. For more information, refer to Presence Expressions.

Amazon social identity provider

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

> **Note**
>
> Amazon as a social identity provider requires access over secure HTTP (HTTPS).

Set up Amazon

To set up Amazon as a social identity provider, first Register for Login With Amazon. You will need an Amazon account.

Then, create a security profile. You will need the following information:

- Security Profile Name (The name of your app)
- Security Profile Description
- Consent Privacy Notice URL
- Consent Logo Image (optional)

When complete and saved, you should see a list of security profiles with `OAuth2` credentials. You should be able to find the `Client ID` and `Client Secret` from this screen.

You still need to configure the web settings for your new Security Profile. From the Amazon Developer Console dashboard, select **Apps and Services > Login with Amazon**, then select **Manage** > **Web Settings**.

In the `Web Settings` for your app, you'll need to set either of the following properties:

- Allowed Origins, which should match the URL for your registration page, such as `https://openidm.example.com:8443`
- Allowed Return URLs, which should match the redirect URIs described in Configure an Amazon Social Identity Provider. You may refer to URIs such as `https://openidm.example.com:8443/` .

Configure an Amazon social identity provider

To configure an Amazon social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.
2. On the **Social Identity Providers** page, enable **Amazon**.
3. In the **Amazon Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to Amazon Social Identity Provider Configuration Details.

After you save the social identity provider configuration, IDM generates a `conf/identityProvider-amazon.json` file:

```
{
   "provider" : "amazon",
   "authorizationEndpoint" : "https://www.amazon.com/ap/oa",
   "tokenEndpoint" : "https://api.amazon.com/auth/o2/token",
   "userInfoEndpoint" : "https://api.amazon.com/user/profile"
   "enabled" : true,
   "clientId" : "<someUUID>",
   "clientSecret" : {
       "$crypto" : {
           "type" : "x-simple-encryption",
           "value" : {
               "cipher" : "AES/CBC/PKCS5Padding",
               "stableId" : "openidm-sym-default",
               "salt" : "<hashValue>",
               "data" : "<encryptedValue>",
               "keySize" : 16,
               "purpose" : "idm.config.encryption",
               "iv" : "<encryptedValue>",
               "mac" : "<hashValue>"
           }
       }
   },
   "scope" : [
       "profile"
   ],
...
```

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the admin UI. When you've registered a user with an Amazon social identity, you can verify this by selecting **Manage > Amazon**, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to Amazon Social Identity Provider Configuration Details.

Configure user registration to link to Amazon

Once you've configured the Amazon social identity provider, you can activate it through User Registration. To do so in the admin UI, select **Configure > User Registration**, and activate that feature. Under the **Social** tab that appears, enable **Social Registration**. For more information on IDM user self-service features, refer to IDM user interface.

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Amazon social identity provider configuration details

You can set up the Amazon social identity provider through the admin UI or in a `conf/identityProvider-amazon.json` file. IDM generates the `identityProvider-amazon.json` file when you configure and enable this social identity provider in the admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the admin UI Amazon Provider pop-up window, along with associated information in the `identityProvider-amazon.json` file:

Amazon social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your Amazon App
Client Secret	`clientSecret`	Used with the Client ID to access the applicable Amazon API
Scope	`scope`	An array of strings that allows access to user data; refer to Amazon's Customer Profile Documentation.
Authorization Endpoint	`authorizationEndpoint`	Typically `https://www.amazon.com/ap/oa`.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://api.amazon.com/auth/o2/token`
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; typically `https://api.amazon.com/user/profile`
Not in the admin UI	`name`	Name of the social identity provider
Not in the admin UI	`type`	Authentication module
Not in the admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider
Not in the admin UI	`propertyMap`	Mapping between Amazon and IDM

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

Apple social identity provider

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

To configure Apple as a social identity provider (Sign in with Apple), you'll need an Apple developer account.

Configure Apple Login

You need a client ID and client secret for your application. In the Apple developer portal, the client ID is called a `Services ID`.

1. Log in to the Apple Developer Portal.
2. Select **Certificates, Identifiers and Profiles > Identifiers**.
3. On the **Identifiers** page, select **Register a New Identifier**, then select **Services IDs**.

4. Enter a **Description** and **Identifier** for this Services ID, and make sure that **Sign in With Apple** is enabled.

   > **Important**
   >
   > The Identifier you specify here will be your **OAuth Client ID**.

5. Click **Configure**.

6. On the **Web Authentication Configuration** screen, enter the **Web Domain** on which IDM runs, and specify the redirect URL used during the OAuth flow (**Return URLs**).

   The redirect URL must have the following format:

   ```
   https://idm.example.com/redirect
   ```

   > **Note**
   >
   > You must use a real domain (FQDN). Apple does not allow `localhost` URLs. If you enter an IP address such as `127.0.0.1`, it will fail later in the OAuth flow.

7. Click **Save > Continue > Register**.

8. Generate the client secret.

   Instead of using simple strings as OAuth client secrets, Apple uses a public/private key pair, where the client secret is a signed JWT. To register the private key with Apple:

   - Select **Certificates, Identifiers and Profiles > Keys**, then click the **+** button to register a new key.
   - Enter a **Key Name**, and enable **Sign In with Apple**.
   - Click **Configure**, and select the primary **App ID** that you created previously.
   - Apple generates a new private key, in a `.p8` file.

     > **Caution**
     >
     > You can only download this key *once*. Ensure that you save this file, because you will not be able to download it again.

     Rename the file to `key.txt`, then locate the Key ID in that file.

   - Use this private key to generate a client secret JWT. Sign the JWT with your private key, using an ES256 algorithm.

Configure an Apple identity provider

To configure an Apple social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.

2. On the **Social Identity Providers** page, enable **Apple**.

3. In the **Apple Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to Apple Social Identity Provider Configuration Details.

Configure user registration through Apple

To configure Apple social user registration using the admin UI:

1. From the navigation bar, click **Configure > User Registration**, and click the **Social** tab.
2. Enable **Social Registration**.

   For more information, refer to Self-service end user UI.

Apple social identity provider configuration details

You can set up the Apple social identity provider through the admin UI or in a `conf/identityProvider-apple.json` file. IDM generates the `identityProvider-apple.json` file when you configure and enable this social identity provider in the admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the admin UI Apple Provider pop-up window, along with associated information in the `identityProvider-apple.json` file.

Apple social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your Apple App. In the Apple developer portal, the client ID is called a `Services ID`.
Client Secret	`clientSecret`	Used with the Client ID to access the applicable Apple API.
Scope	`scope`	An array of strings that allows access to user data.
Authorization Endpoint	`authorizationEndpoint`	Typically, `https://appleid.apple.com/auth/authorize`.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token. Typically, `https://appleid.apple.com/auth/token`.
Well-Known Endpoint	`wellKnownEndpoint`	Access for other URIs. Typically, `https://appleid.apple.com/.well-known/openid-configuration`.
Issuer	`issuer`	The token issuer. Typically, `https://appleid.apple.com`.
Not in the admin UI	`provider`	Name of the social identity provider.
Not in the admin UI	`configClass`	Configuration class for the authentication module.
Not in the admin UI	`basicAuth`	Whether to use basic authentication.
Not in the admin UI	`propertyMap`	Mapping between Apple and IDM.

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

Facebook social identity provider

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

> **Note**
>
> Facebook as a social identity provider requires access over secure HTTP (HTTPS).

Set up Facebook

To set up Facebook as a social identity provider, navigate to the Facebook for Developers page. You'll need a Facebook account. While you could use a personal Facebook account, it is best to use an organizational account to avoid problems if specific individuals leave your organization. When you set up a Facebook social identity provider, you'll need to perform the following tasks:

- On the **Facebook for Developers** page, select **My Apps**, and click **Add a New App**. For IDM, you'll create a **Website** application.
- You'll need to include the following information when creating a Facebook website application:
  - **Display Name**
  - **Contact Email**
  - **IDM URL**
- When complete, you should see your App. Navigate to **Basic Settings**.
- Make a copy of the **App ID** and **App Secret** for when you configure the Facebook social identity provider in IDM.
- In App settings, you should see an entry for **App Domains**, such as `example.com`, as well as a **Website Site URL**, such as `https://idm.example.com/`.

For Facebook's documentation on the subject, refer to Facebook Login for the Web with the JavaScript SDK.

Configure a Facebook social identity provider

To configure a Facebook social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.
2. On the **Social Identity Providers** page, enable **Facebook**.
3. In the **Facebook Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to Facebook Social Identity Provider Configuration Details.

After you save the social identity provider configuration, IDM generates a `conf/identityProvider-facebook.json` file:

```
{
   "provider" : "facebook",
   "authorizationEndpoint" : "https://www.facebook.com/dialog/oauth",
   "tokenEndpoint" : "https://graph.facebook.com/v2.7/oauth/access_token",
   "userInfoEndpoint" : "https://graph.facebook.com/me?fields=id,name,picture,email,first_name,last_name,locale"
   "clientId" : "<someUUID>",
   "clientSecret" : {
       "$crypto" : {
           "type" : "x-simple-encryption",
           "value" : {
             "cipher" : "AES/CBC/PKCS5Padding",
             "stableId" : "openidm-sym-default",
             "salt" : "<hashValue>",
             "data" : "<encryptedValue>",
             "keySize" : 16,
             "purpose" : "idm.config.encryption",
             "iv" : "<encryptedValue>",
             "mac" : "<hashValue>"
           }
       }
   },
   "scope" : [
       "email",
       "user_birthday"
   ],
...
```

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the admin UI. When you've registered a user with a Facebook social identity, you can verify this by selecting **Manage > Facebook**, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: Facebook Social Identity Provider Configuration Details.

Configure user registration to link to Facebook

Once you've configured the Facebook social identity provider, you can activate it through User Registration. To do so in the admin UI, select Configure > User Registration, and under the Social tab, enable the option associated with Social Registration. For more information about user self-service features, refer to Self-service end user UI.

When you enable social registration, you're allowing users to register on IDM through all active social identity providers.

Facebook social identity provider configuration details

You can set up the Facebook social identity provider through the admin UI or in a `conf/identityProvider-facebook.json` file. IDM generates the `identityProvider-facebook.json` file when you configure and enable this social identity provider in the admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the admin UI Facebook Provider pop-up window, along with associated information in the `identityProvider-facebook.json` file:

Facebook social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
App ID	`clientId`	The client identifier for your Facebook App
App Secret	`clientSecret`	Used with the App ID to access the applicable Facebook API
Scope	`scope`	An array of strings that allows access to user data; refer to Facebook's Permissions Reference Documentation.
Authorization Endpoint	`authorizationEndpoint`	For Facebook's implementation, refer to their documentation on how they Manually Build a Login Flow.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token. For Facebook's implementation, refer to their documentation on how they Manually Build a Login Flow.
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields through Facebook's API. The default endpoint includes the noted field properties as a list, as defined in Facebook's Permissions Reference.
Not in the admin UI	`name`	Name of the Social ID provider
Not in the admin UI	`type`	Authentication module
Not in the admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider
Not in the admin UI	`propertyMap`	Mapping between Facebook and IDM

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

Google social identity provider

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

Set up Google

To set up Google as a social identity provider, navigate to the Google API Manager. You'll need a Google account. If you have Gmail, you already have a Google account ☺. While you could use a personal Google account, it is best to use an organizational account to avoid problems if specific individuals leave your organization. When you set up a Google social identity provider, you'll need to perform the following tasks:

Plan ahead. It may take some time before the Google+ API that you configure for IDM is ready for use.

- In the **Google API Manager**, select and enable the **Google+** API. It is one of the Google "social" APIs.
- Create a project for IDM.
- Create OAuth client ID credentials. You'll need to configure an `OAuth consent screen` with at least a product name and email address.
- When you set up a Web application for the client ID, you'll need to set up a web client with:
  - `Authorized JavaScript origins`

    The origin URL for IDM, typically a URL such as `https://openidm.example.com:8443`
  - `Authorized redirect URIs`

    The redirect URI after users are authenticated, typically, `https://openidm.example.com:8443/`
- In the list of credentials, you'll refer to a unique `Client ID` and `Client secret`. You'll need this information when you configure the Google social identity provider, as described in Configure a Google Social Identity Provider.

For Google's procedure, refer to the Google Identity Platform documentation on Setting Up OAuth 2.0.

Configure a Google social identity provider

To configure a Google social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.
2. On the **Social Identity Providers** page, enable **Google**.
3. In the **Google Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to Google Social Identity Provider Configuration Details.

After you save the social identity provider configuration, IDM generates a `conf/identityProvider-google.json` file:

```
{
    "enabled" : true,
    "authorizationEndpoint" : "https://accounts.google.com/o/oauth2/v2/auth",
    "tokenEndpoint" : "https://www.googleapis.com/oauth2/v4/token",
    "userInfoEndpoint" : "https://www.googleapis.com/oauth2/v3/userinfo",
    "wellKnownEndpoint" : "https://accounts.google.com/.well-known/openid-configuration",
    "issuer": "https://accounts.google.com",
    "clientId" : "<someUUID>",
    "clientSecret" : {encrypted-client-secret},
...
```

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the admin UI. When you've registered a user with a Google social identity, you can verify this by selecting **Manage > Google**, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: Google Social Identity Provider Configuration Details.

Configure user registration to link to Google

Once you've configured the Google social identity provider, you can activate it through User Registration. To do so in the admin UI, select **Configure > User Registration**, and under the **Social** tab, enable the option associated with Social Registration. For more information on user self-service features, refer to Self-service end user UI.

When you enable social registration, you're allowing users to register on IDM through all active social identity providers.

Google social identity provider configuration details

You can set up the Google social identity provider through the admin UI or in a `conf/identityProvider-google.json` file. IDM generates the `identityProvider-google.json` file when you configure and enable this social identity provider in the admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the admin UI Google Provider pop-up window, along with associated information in the `identityProvider-google.json` file:

Google social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your Google Identity Platform project.
Client Secret	`clientSecret`	Used with the Client ID to access the configured Google API.
Scope	`scope`	An array of strings that allows access to user data; refer to Google's documentation on Authorization Scopes.
Authorization Endpoint	`authorizationEndpoint`	As per RFC 6749, "used to interact with the resource owner and obtain an authorization grant". For Google's implementation, refer to Forming the URL.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization grant, and returns an access and ID token.
User Info Endpoint	`userInfoEndpoint`	Endpoint that receives an access token, and returns information about the user.
Well-Known Endpoint	`wellKnownEndpoint`	Access URL for Google's Discovery Document.
Issuer	`issuer`	The token issuer. Typically, `https://accounts.google.com`.
Not in the admin UI	`name`	Name of the social identity provider.
Not in the admin UI	`type`	Authentication module.

Property (UI)	Property (JSON file)	Description
Not in the admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider.
Not in the admin UI	`propertyMap`	Mapping between Google and IDM.

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

Instagram social identity provider

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

Set up Instagram

To set up Instagram as a social identity provider, navigate to Facebook for Developers, and follow the steps. You'll need a minimum of:

- An Instagram account
- A Facebook developer account
- An application name and description
- A website URL for your app, such as `http://openidm.example.com:8080`
- A Redirect URL for IDM, such as `http://openidm.example.com:8080/`

Configure an Instagram social identity provider

To configure an Instagram social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.
2. On the **Social Identity Providers** page, enable **Instagram**.
3. In the **Instagram Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to Instagram Social Identity Provider Configuration Details.

After you save the social identity provider configuration, IDM generates a `conf/identityProvider-instagram.json` file:

```
{
   "provider" : "instagram",
   ...
   "clientId" : "<Client_ID_Name>",
   "clientSecret" : {
      "$crypto" : {
         "type" : "x-simple-encryption",
         "value" : {
            "cipher" : "AES/CBC/PKCS5Padding",
            "stableId" : "openidm-sym-default",
            "salt" : "<hashValue>",
            "data" : "<encryptedValue>",
            "keySize" : 16,
            "purpose" : "idm.config.encryption",
            "iv" : "<encryptedValue>",
            "mac" : "<hashValue>"
         }
      }
   },
   "authorizationEndpoint" : "https://api.instagram.com/oauth/authorize/",
   "tokenEndpoint" : "https://api.instagram.com/oauth/access_token",
   "userInfoEndpoint" : "https://graph.instagram.com/me?fields=id,username",
   "redirectUri" : "http://openidm.example.com:8080/",
   "scope" : [
       "user_profile",
   ],
...
```

The file includes `schema` information for each social identity account, as collected by IDM, as well as the order in which it appears in the admin UI. When you've registered a user with an Instagram social identity, you can verify this by selecting **Manage > Instagram**, and then selecting a user. For more information about the properties in this file, refer to Instagram Social Identity Provider Configuration Details.

Configure user registration to link to Instagram

After you configure the Instagram social identity provider, you can activate it through User Registration. To do so in the admin UI, select **Configure > User Registration**, and activate that feature. Under the **Social** tab that appears, enable **Social Registration**. For more information on IDM user self-service features, refer to Self-service end user UI.

> **Note**
>
> When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Instagram social identity provider configuration details

You can set up the Instagram social identity provider through the admin UI or in a `conf/identityProvider-instagram.json` file. IDM generates the `identityProvider-instagram.json` file when you configure and enable this social identity provider in the admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the admin UI Instagram Provider pop-up window, along with associated information in the `identityProvider-instagram.json` file:

Instagram social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	Your Instagram App client identifier
Client Secret	`clientSecret`	Used with the Client ID to access the Instagram API
Scope	`scope`	An array of strings that allows access to user data
Authorization Endpoint	`authorizationEndpoint`	Typically `https://api.instagram.com/oauth/authorize/` ; known as an Instagram *Authorize URL*
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://api.instagram.com/oauth/access_token`
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; typically `https://graph.instagram.com/me?fields=id,username`
Not in the admin UI	`provider`	Name of the social identity provider
Not in the admin UI	`configClass`	Configuration class for the authentication module
Not in the admin UI	`basicAuth`	Whether to use basic authentication
Not in the admin UI	`propertyMap`	Mapping between Instagram and IDM

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

LinkedIn social identity provider

Important

Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

Important

Microsoft has deprecated the "Sign In with LinkedIn" functionality as of August 1, 2023. Refer to Sign In with LinkedIn.

Set up a LinkedIn app

Before you start, you will need a LinkedIn account. You can use a personal LinkedIn account for testing, but you should use an organizational account to avoid problems if individuals leave your organization.

To set up a LinkedIn app:

1. Log in to LinkedIn, and navigate to LinkedIn Developers → MyApps.
2. Select **Create app**, and enter the following information:
    - **App name**—Any unique name fewer than 50 characters.
    - **Company**—The company name associated with this application.
    - **Privacy policy URL**—An optional URL that displays a privacy policy.
    - **Business email**—The business email address that is associated with this application.
    - **App logo**—The logo that is displayed to users when they authenticate with this app.
3. Select the products that should be integrated into the app.
4. **Accept** LinkedIn's legal terms.
5. Select **Verify to associate the app with your company**, then follow the verification approval process.
6. After you have approved the app, select it under **My Apps**, then select the **Auth** tab.
7. Take note of the `Client ID` and `Client Secret` —you will need them in the next procedure.
8. The app should have the following Permissions:
    - `r_emailaddress`
    - `r_liteprofile`
    - `w_member_social`
9. Under OAuth 2.0 settings, select **Add redirect URL** and enter the FQDN and port number of your IDM instance. For example, `http://openidm.example.com:8080/`

> **Note**
>
> For LinkedIn's procedure, refer to their documentation on Authenticating with OAuth 2.0.

Configure a LinkedIn social identity provider

To configure a LinkedIn social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.
2. On the **Social Identity Providers** page, enable **LinkedIn**.
3. In the **LinkedIn Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to LinkedIn Social Identity Provider Configuration Details.

After you save the social identity provider configuration, IDM generates a `conf/identityProvider-linkedIn.json` file:

```
{
    "provider" : "linkedIn",
    "authorizationEndpoint" : "https://www.linkedin.com/oauth/v2/authorization",
    "tokenEndpoint" : "https://www.linkedin.com/oauth/v2/accessToken",
    "userInfoEndpoint" : "https://api.linkedin.com/v2/me?
projection=(id,firstName,lastName,profilePicture(displayImage~:playableStreams))",
    "emailAddressEndpoint" : "https://api.linkedin.com/v2/emailAddress?q=members&projection=(elements*(handle~))",
    "clientId" : "77l9udb8qmqihq",
    "clientSecret" : {
        "$crypto" : {
            "type" : "x-simple-encryption",
            "value" : {
                "cipher" : "AES/CBC/PKCS5Padding",
                "stableId" : "openidm-sym-default",
                "salt" : "2cmC36Ds++6xAtRhlvNOEw==",
                "data" : "TJ7VOHjJI0VWWedTKX4agviqc3H3Un5RDVAWyB2u64g=",
                "keySize" : 16,
                "purpose" : "idm.config.encryption",
                "iv" : "QbGAUSuOMrCh1i8F0fWGyA==",
                "mac" : "rUFVcSJ5+s+LZL6YFB3rFQ=="
            }
        }
    },
    "scope" : [
        "r_liteprofile",
        "r_emailaddress"
    ],
...
```

The file includes `schema` information, indicating the properties of each social identity account that will be collected by IDM, and the order in which these properties appear in the admin UI. When you have registered a user with a LinkedIn social identity, you can verify these properties by selecting **Manage > LinkedIn**, then selecting the user.

Further down in the file, the `propertyMap` maps user information between the `source` (social identity provider) and the `target` (IDM).

For more information about the properties in this file, refer to LinkedIn Social Identity Provider Configuration Details.

Configure user registration with LinkedIn

To configure LinkedIn social user registration using the admin UI:

1. From the navigation bar, click **Configure > User Registration**, and click the **Social** tab.
2. Enable **Social Registration**.

   For more information, refer to Self-service end user UI.

> **Note**
>
> When you enable social registration, you are allowing users to register in IDM through all active social identity providers.

LinkedIn social identity provider configuration details

You can set up the LinkedIn social identity provider through the admin UI or in a `conf/identityProvider-linkedIn.json` file. IDM generates the `identityProvider-linkedIn.json` file when you configure and enable this social identity provider in the admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the admin UI LinkedIn Provider pop-up window, along with associated information in the `identityProvider-linkedIn.json` file:

LinkedIn social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your LinkedIn Application
Client Secret	`clientSecret`	Used with the Client ID to access the applicable LinkedIn API
Scope	`scope`	An array of strings that allows access to user data; refer to LinkedIn's documentation on Lite Profile Fields.
Authorization Endpoint	`authorizationEndpoint`	As per RFC 6749, "used to interact with the resource owner and obtain an authorization grant". For LinkedIn's implementation, refer to their documentation on Authenticating with OAuth 2.0.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token. For LinkedIn's implementation, refer to their documentation on Authenticating with OAuth 2.0.
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields through LinkedIn's API.
Email Address Endpoint	`emailAddressEndpoint`	API that must be called to retrieve the email address of the user.
Well-Known Endpoint	`wellKnownEndpoint`	Not used for LinkedIn
Not in the admin UI	`name`	Name of the social identity provider
Not in the admin UI	`type`	Authentication module
Not in the admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider
Not in the admin UI	`propertyMap`	Mapping between LinkedIn and IDM

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

Microsoft social identity provider

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

> **Note**
>
> Microsoft as a social identity provider requires access over secure HTTP (HTTPS). This example assumes that you've configured IDM on `https://openidm.example.com:8443`. Substitute your URL for `openidm.example.com`.

Set up Microsoft

For Microsoft documentation on how to set up a social identity provider, navigate to the following article: Sign-in Microsoft Account & Azure AD users in a single app. You'll need a Microsoft account.

To set up Microsoft as a social identity provider:

1. Navigate to the Microsoft app registration portal, and sign in with your Microsoft account.
2. Select **Add an App**, and give your app a name.

   The portal will assign your app a unique `Application ID`.
3. To find your *Application Secret*, select **Generate New Password**. The displayed password is your *Application Secret*.

   > **Tip**
   >
   > Store the *Application Secret* in a secure location, as you can't view it again.
4. Select **Add Platform**, and enter the following details:
   - **Web** platform.
   - Enable **Allow Implicit Flow**
   - **Redirect URI**: `https://openidm.example.com:8443/`
   - **Logo image** (optional)
   - **Terms of Service URL** (optional)
   - **Privacy Statement URL** (optional)

The `OAuth2` credentials for your new Microsoft App include an `Application ID` and `Application Secret` for your app.

Configure a Microsoft social identity provider

To configure a Microsoft social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.
2. On the **Social Identity Providers** page, enable **Microsoft**.

3. In the **Microsoft Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to Microsoft Social Identity Provider Configuration Details.

After you save the social identity provider configuration, IDM generates a `conf/identityProvider-microsoft.json` file:

```
"provider" : "microsoft",
   "authorizationEndpoint" : "https://login.microsoftonline.com/common/oauth2/v2.0/authorize",
   "tokenEndpoint" : "https://login.microsoftonline.com/common/oauth2/v2.0/token",
   "userInfoEndpoint" : "https://graph.microsoft.com/v1.0/me"
   "clientId" : "<someUUID>",
   "clientSecret" : {
       "$crypto" : {
           "type" : "x-simple-encryption",
           "value" : {
               "cipher" : "AES/CBC/PKCS5Padding",
               "stableId" : "openidm-sym-default",
               "salt" : "<hashValue>",
               "data" : "<encryptedValue>",
               "keySize" : 16,
               "purpose" : "idm.config.encryption",
               "iv" : "<encryptedValue>",
               "mac" : "<hashValue>"
           }
       }
   },
   "scope" : [
       "User.Read"
   ],
...
```

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the admin UI. When you've registered a user with a Microsoft social identity, you can verify this by selecting **Manage > Microsoft**, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: Microsoft Social Identity Provider Configuration Details.

Configure user registration to link to Microsoft

Once you've configured the Microsoft social identity provider, you can activate it through User Registration. To do so in the admin UI, select **Configure > User Registration**, and activate that feature. Under the **Social** tab that appears, enable **Social Registration**. For more information on IDM user self-service features, refer to Self-service end user UI.

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Microsoft social identity provider configuration details

You can set up the Microsoft social identity provider through the admin UI or in a `conf/identityProvider-microsoft.json` file. IDM generates the `identityProvider-microsoft.json` file when you configure and enable this social identity provider in the admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the admin UI Microsoft Provider pop-up window, along with associated information in the `identityProvider-microsoft.json` file:

Microsoft social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Application ID	`clientId`	The client identifier for your Microsoft App
Application Secret	`clientSecret`	Used with the Application ID; shown as application password
Scope	`scope`	OAuth 2 scopes; for more information, refer to Microsoft Graph Permission Scopes.
Authorization Endpoint	`authorizationEndpoint`	Typically `https://login.microsoftonline.com/common/oauth2/v2.0/authorize`
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code and returns an access token; typically `https://login.microsoftonline.com/common/oauth2/v2.0/token`
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; typically `https://graph.microsoft.com/v1.0/me`
Not in the admin UI	`name`	Name of the social identity provider
Not in the admin UI	`type`	Authentication module
Not in the admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider
Not in the admin UI	`propertyMap`	Mapping between Microsoft and IDM

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

Salesforce social identity provider

Important

Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

Note

When you configure a Salesforce app, look for a *Consumer Key* and a *Consumer Secret*. IDM uses this information as a `clientId` and `clientSecret`, respectively.
For reference, read through the following Salesforce documentation: Connected Apps Overview.

Set up Salesforce

> **Note**
>
> These instructions were written with the Winter '19 Release of the Salesforce API. The menu items might differ slightly if you are working with a different version of the API.

1. To set up Salesforce as a social identity provider, you will need a Salesforce developer account. Log in to the Salesforce Developers Page with your developer account credentials and create a new **Connected App**.
2. Under **App Setup**, select **Create > Apps > Connected Apps > New**. You will need to add the following information:
   - **Connected App Name**
   - **API Name** (defaults to the **Connected App Name**)
   - **Contact Email**
   - Activate **Enable OAuth Settings**
   - **Callback URL** (also known as the *Redirect URI* for other providers), for example `https://localhost:8443`.

     The Callback URL must correspond to the log-in URL for the IDM admin UI.
3. Add the following OAuth scopes:
   - **Access and Manage your data (api)**
   - **Access your basic information (id, profile, email, address, phone)**
   - **Perform requests on your behalf at any time (refresh_token, offline_access)**
   - **Provide access to your data via the Web (web)**

     > **Note**
     >
     > You must add these scopes even if you are planning to use the `full` OAuth scope.
4. After you have saved the **Connected App**, it might take a few minutes for the new app to appear under **Administration Setup > Manage Apps > Connected Apps**.
5. Select the new **Connected App** then locate the **Consumer Key** and **Consumer Secret** (under the API list). You'll use that information as shown here:
   - **Salesforce Consumer Key** = *IDM Client ID*
   - **Salesforce Consumer Secret** = *IDM Client Secret*

Configure a Salesforce social identity provider

To configure a Salesforce social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.
2. On the **Social Identity Providers** page, enable **Salesforce**.

3. In the **Salesforce Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to Salesforce Social Identity Provider Configuration Details.

After you save the social identity provider configuration, IDM generates a `conf/identityProvider-salesforce.json` file:

```
{
    "provider" : "salesforce",
    "authorizationEndpoint" : "https://login.salesforce.com/services/oauth2/authorize",
    "tokenEndpoint" : "https://login.salesforce.com/services/oauth2/token",
    "userInfoEndpoint" : "https://login.salesforce.com/services/oauth2/userinfo",
    "clientId" : "<someUUID>",
    "clientSecret" : {
       "$crypto" : {
           "type" : "x-simple-encryption",
           "value" : {
               "cipher" : "AES/CBC/PKCS5Padding",
               "stableId" : "openidm-sym-default",
               "salt" : "<hashValue>",
               "data" : "<encryptedValue>",
               "keySize" : 16,
               "purpose" : "idm.config.encryption",
               "iv" : "<encryptedValue>",
               "mac" : "<hashValue>"
           }
       }
    },
    "scope" : [
        "id",
        "api",
        "web"
    ],
```

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the admin UI. When you've registered a user with a Salesforce social identity, you can verify this by selecting **Manage > Salesforce**, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: Salesforce Social Identity Provider Configuration Details.

Configure user registration to link to Salesforce

Once you've configured the Salesforce social identity provider, you can activate it through User Registration. To do so in the admin UI, select **Configure > User Registration**, and activate that feature. Under the **Social** tab that appears, enable **Social Registration**. For more information on IDM user self-service features, refer to Self-service end user UI.

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Salesforce social identity provider configuration details

You can set up the Salesforce social identity provider through the admin UI or in a `conf/identityProvider-salesforce.json` file. IDM generates the `identityProvider-salesforce.json` file when you configure and enable this social identity provider in the admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the admin UI Salesforce Provider pop-up window, along with associated information in the `identityProvider-salesforce.json` file:

Salesforce social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your Salesforce App
Client Secret	`clientSecret`	Used with the Client ID to access the applicable Salesforce API
Scope	`scope`	An array of strings that allows access to user data
Authorization Endpoint	`authorizationEndpoint`	A typical URL: `https://login.salesforce.com/services/oauth2/authorize`.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; such as `https://login.salesforce.com/services/oauth2/token`
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; a typical URL: `https://login.salesforce.com/services/oauth2/userinfo`
Not in the admin UI	`provider`	Name of the social identity provider
Not in the admin UI	`configClass`	Configuration class for the authentication module
Not in the admin UI	`basicAuth`	Whether to use basic authentication
Not in the admin UI	`propertyMap`	Mapping between Salesforce and IDM

Twitter social identity provider

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

Set up Twitter

For additional information, refer to Single-user OAuth with Examples.

1. To set up Twitter as a social identity provider, you'll need a Twitter account, and then navigate to Twitter Application Management.
2. Select **Create New App**, and enter at least the following information:
    - **Name**

- **Description**
- **Website**, such as `http://openidm.example.com:8080`
- **Callback URL**, such as `http://openidm.example.com:8080/` ; required for IDM; for other providers, known as `RedirectURI`

3. Click **Save**.

   The page displays a `Consumer Key` and `Consumer Secret` for your new web app.

> **Note**
>
> Twitter Apps use the OAuth 1.0a protocol. With IDM, you can use the same process used to configure OIDC and OAuth 2 social identity providers.

Configure Twitter as a social identity provider

To configure a Twitter social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.
2. On the **Social Identity Providers** page, enable **Twitter**.
3. In the **Twitter Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to Twitter Social Identity Provider Configuration Details.

After you save the social identity provider configuration, IDM generates a `conf/identityProvider-twitter.json` file:

```
{
    "provider" : "twitter",
    "requestTokenEndpoint" : "https://api.twitter.com/oauth/request_token",
    "authorizationEndpoint" : "https://api.twitter.com/oauth/authenticate",
    "tokenEndpoint" : "https://api.twitter.com/oauth/access_token",
    "userInfoEndpoint" : "https://api.twitter.com/1.1/account/verify_credentials.json",
    "clientId" : "<Client_ID_Name>",
    "clientSecret" : {
      "$crypto" : {
          "type" : "x-simple-encryption",
          "value" : {
              "cipher" : "AES/CBC/PKCS5Padding",
              "stableId" : "openidm-sym-default",
              "salt" : "<hashValue>",
              "data" : "<encryptedValue>",
              "keySize" : 16,
              "purpose" : "idm.config.encryption",
              "iv" : "<encryptedValue>",
              "mac" : "<hashValue>"
          }
      }
    },
```

The next part of the file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the admin UI. When you've registered a user with a Twitter social identity, you can verify this by selecting **Manage > Twitter**, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: Twitter Social Identity Provider Configuration Details.

Configure user registration to link to Twitter

Once you've configured the Twitter social identity provider, you can activate it through User Registration. To do so in the admin UI, select **Configure > User Registration**, and activate that feature. Under the **Social** tab that appears, enable **Social Registration**. For more information on IDM user self-service features, refer to Self-service end user UI.

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Twitter social identity provider configuration details

You can set up the Twitter social identity provider through the admin UI or in a `conf/identityProvider-twitter.json` file. IDM generates the `identityProvider-twitter.json` file when you configure and enable the Twitter social identity provider in the admin UI. Alternatively, you can create that file manually.

The following table includes the information shown in the admin UI Twitter Provider pop-up window, along with associated information in the `identityProvider-twitter.json` file.

Twitter social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Consumer Key	`clientId`	The client identifier for your Twitter App
Consumer Secret	`clientSecret`	Used with the Client ID to access the applicable Twitter API
Authorization Endpoint	`authorizationEndpoint`	Typically `https://api.twitter.com/oauth/authenticate`; known as a Twitter *Authorize URL*
Access Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://api.twitter.com/oauth/access_token`
User Info Endpoint	`userInfoEndpoint`	Access for other URIs; typically `https://api.twitter.com/1.1/account/verify_credentials.json`
Request Token Endpoint	`requestTokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://api.twitter.com/oauth/request_token`
Not in the admin UI	`provider`	Name of the social identity provider
Not in the admin UI	`authenticationIdKey`	The user identity property, such as `_id`

Property (UI)	Property (JSON file)	Description
Not in the admin UI	`configClass`	Configuration class for the authentication module
Not in the admin UI	`basicAuth`	Whether to use basic authentication
Not in the admin UI	`propertyMap`	Mapping between Twitter and IDM

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

Vkontakte social identity provider

Important

Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

Note

When you configure a Vkontakte app, look for an *Application ID* and a *Secure Key*. IDM uses this information as a `clientId` and `clientSecret`, respectively.

Set up Vkontakte

1. To set up Vkontakte as a social identity provider, navigate to the Vkontakte Developers Page. You'll need a Vkontakte account.
2. Click **My Apps**, and create an application with the following information:
   - **Title** (The name of your app)
   - **Platform** (Choose **Website**)
   - **Site Address** (The URL of your IDM deployment, such as `http://openidm.example.com:8080/`
   - **Base domain** (Example: `example.com`)
   - **Authorized Redirect URI** (Example: `http://openidm.example.com:8080/`)
   - **API Version**; for the current VKontakte API version, refer to VK Developers Documentation, API Versions. The default VKontakte API version used for IDM 8.0 is 5.73.

   Note

   If you leave and need to return to Vkontakte, navigate to `https://vk.com/dev` and select **My Apps**. You can then **Manage** the new apps that you've created.
3. Navigate to the **Settings** for your app, where you'll find the *Application ID* and *Secure Key*. You'll use that information as shown here:
   - **Vkontakte Application ID** = *IDM Client ID*

- **Vkontakte Secure Key** = *IDM Client Secret*

Configure a Vkontakte social identity provider

To configure a Vkontakte social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.
2. On the **Social Identity Providers** page, enable **Vkontakte**.
3. In the **Vkontakte Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to Vkontakte Social Identity Provider Configuration Details.

After you save the social identity provider configuration, IDM generates a `conf/identityProvider-vkontakte.json` file:

```
{
    "provider" : "vkontakte",
    "configClass" : "org.forgerock.oauth.clients.vk.VKClientConfiguration",
    "basicAuth" : false,
    "clientId" : "<someUUID>",
    "clientSecret" : {
        "$crypto" : {
            "type" : "x-simple-encryption",
            "value" : {
                "cipher" : "AES/CBC/PKCS5Padding",
                "stableId" : "openidm-sym-default",
                "salt" : "<hashValue>",
                "data" : "<encryptedValue>",
                "keySize" : 16,
                "purpose" : "idm.config.encryption",
                "iv" : "<encryptedValue>",
                "mac" : "<hashValue>"
            }
        }
    },
    "authorizationEndpoint" : "https://oauth.vk.com/authorize",
    "tokenEndpoint" : "https://oauth.vk.com/access_token",
    "userInfoEndpoint" : "https://api.vk.com/method/users.get",
    "redirectUri" : "http://openidm.example.com:8080/",
    "apiVersion" : "5.73",
    "scope" : [
        "email"
    ],
...
```

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the admin UI. When you've registered a user with a Vkontakte social identity, you can verify this by selecting **Manage > Vkontakte**, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: Vkontakte Social Identity Provider Configuration Details.

Configure user registration to link to Vkontakte

Once you've configured the Vkontakte social identity provider, you can activate it through User Registration. To do so in the admin UI, select **Configure > User Registration**, and activate that feature. Under the **Social** tab that appears, enable **Social Registration**. For more information on IDM user self-service features, refer to Self-service end user UI.

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Vkontakte social identity provider configuration details

You can set up the Vkontakte social identity provider through the admin UI or in a `conf/identityProvider-vkontakte.json` file. IDM generates the `identityProvider-vkontakte.json` file when you configure and enable this social identity provider in the admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the admin UI Vkontakte Provider pop-up window, along with associated information in the `identityProvider-vkontakte.json` file:

Vkontakte social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Application ID	`clientId`	The client identifier for your Vkontakte App
Secure Key	`clientSecret`	Used with the Client ID to access the applicable Vkontakte API
Scope	`scope`	An array of strings that allows access to user data.
Authorization Endpoint	`authorizationEndpoint`	Typically `https://oauth.vk.com/authorize`.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://oauth.vk.com/access_token`
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; typically `https://api.vk.com/method/users.get`
API Version	`apiVersion`	Version of the applicable VKontakte API, available from VK Developers Documentation, API Versions section. The default VKontakte API version used for IDM 8.0 is 5.73.
Not in the admin UI	`provider`	Name of the social identity provider
Not in the admin UI	`configClass`	Configuration class for the authentication module
Not in the admin UI	`basicAuth`	Whether to use basic authentication
Not in the admin UI	`authenticationIdKey`	The user identity property, such as `id`
Not in the admin UI	`propertyMap`	Mapping between Vkontakte and IDM

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

WeChat social identity provider

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

These procedures assume that you have a WeChat developer account with access to create WeChat web application credentials. To verify access, you'll need the WeChat app on your mobile device.

Set up WeChat

To set up WeChat as a social identity provider, you'll need to get the following information for your WeChat app. The name may be different in WeChat.

- Client ID (WeChat uses `appid` as of this writing.)
- Client Secret (WeChat uses `secret` as of this writing.)
- Scope
- Authorization Endpoint URL
- Token Endpoint URL
- User Info Endpoint URL
- Redirect URI, normally something like `http://openidm.example.com/`

WeChat unique requirements

Before testing WeChat, be prepared for the following special requirements:

- WeChat works only if you deploy IDM on one of the following ports: *80* or *443*.

  For more information on how to configure IDM to use these ports, refer to Host and port information.
- For registration and sign-in, WeChat requires the use of a mobile device with a QR code reader.
- For sign-in, you'll also need to install the WeChat app on your mobile device.

Configure a WeChat social identity provider

To configure a WeChat social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.
2. On the **Social Identity Providers** page, enable **WeChat**.
3. In the **WeChat Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to WeChat Social Identity Provider Configuration Details.

After you save the social identity provider configuration, IDM generates a `conf/identityProvider-wechat.json` file:

```
{
    "provider" : "wechat",
    ...
    "clientId" : "<someUUID>",
    "clientSecret" : {
       "$crypto" : {
           "type" : "x-simple-encryption",
              "value" : {
                "cipher" : "AES/CBC/PKCS5Padding",
                "stableId" : "openidm-sym-default",
                "salt" : "<hashValue>",
                "data" : "<encryptedValue>",
                "keySize" : 16,
                "purpose" : "idm.config.encryption",
                "iv" : "<encryptedValue>",
                "mac" : "<hashValue>"
           }
       }
    },
    "authorizationEndpoint" : "https://open.weixin.qq.com/connect/qrconnect",
    "tokenEndpoint" : "https://api.wechat.com/sns/oauth2/access_token",
    "refreshTokenEndpoint" : "https://api.wechat.com/sns/oauth2/refresh_token",
    "userInfoEndpoint" : "https://api.wechat.com/sns/userinfo",
    "redirectUri" : "http://openidm.example.com:8080/",
    "scope" : [
        "snsapi_login"
    ],
...
```

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the admin UI. When you've registered a user with a WeChat social identity, you can verify this by selecting **Manage > WeChat**, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: WeChat Social Identity Provider Configuration Details.

Configure user registration to link to WeChat

Once you've configured the WeChat social identity provider, you can activate it through User Registration. To do so in the admin UI, select **Configure > User Registration**, and activate that feature. Under the **Social** tab that appears, enable **Social Registration**. For more information on IDM user self-service features, refer to Self-service end user UI.

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

WeChat social identity provider configuration details

You can set up the WeChat social identity provider through the admin UI or in a `conf/identityProvider-wechat.json` file. IDM generates the `identityProvider-wechat.json` file when you configure and enable this social identity provider in the admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the admin UI WeChat Provider pop-up window, along with associated information in the `identityProvider-wechat.json` file.

> **Note**
>
> WeChat supports URLs on one of the following ports: 80 or 443. For more information on how to configure IDM to use these ports, refer to Host and port information.

WeChat social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your WeChat App
Client Secret	`clientSecret`	Used with the Client ID to access the applicable WeChat API
Scope	`scope`	An array of strings that allows access to user data
Authorization Endpoint	`authorizationEndpoint`	Typically `https://open.weixin.qq.com/connect/qrconnect`.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://api.wechat.com/sns/oauth2/access_token`
Refresh Token Endpoint	`refreshTokenEndpoint`	Endpoint that receives a one-time authorization code, and returns a refresh token; typically `https://api.wechat.com/sns/oauth2/refresh_token`
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; typically `https://api.wechat.com/user/profile`
Not in the admin UI	`provider`	Name of the social identity provider
Not in the admin UI	`configClass`	Configuration class for the authentication module
Not in the admin UI	`basicAuth`	Whether to use basic authentication
Not in the admin UI	`propertyMap`	Mapping between WeChat and IDM

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

WordPress social identity provider

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

Set up WordPress

To set up WordPress as a social identity provider, navigate to Developer Resources. You'll need a WordPress account. You can then navigate to the WordPress My Applications page, where you can create a new web application, with the following information:

- **Name**
- **Description**
- **Website URL**, which becomes your Application URL
- **Redirect URL(s)**; for IDM, normally `http://openidm.example.com:8080/`
- **Type**, which allows you to select Web clients

When complete and saved, you should see a list of `OAuth Information` for your new web application. That information should include your `Client ID` and `Client Secret`.

Configure a WordPress social identity provider

To configure a WordPress social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.
2. On the **Social Identity Providers** page, enable **WordPress**.
3. In the **WordPress Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to WordPress Social Identity Provider Configuration Details.

After you save the social identity provider configuration, IDM generates a `conf/identityProvider-wordpress.json` file:

```
{
    "provider" : "wordpress",
    "authorizationEndpoint" : "https://public-api.wordpress.com/oauth2/authorize",
    "tokenEndpoint" : "https://public-api.wordpress.com/oauth2/token",
    "userInfoEndpoint" : "https://public-api.wordpress.com/rest/v1.1/me/",
    "enabled" : true,
    "clientId" : "<someUUID>",
    "clientSecret" : {
       "$crypto" : {
           "type" : "x-simple-encryption",
           "value" : {
               "cipher" : "AES/CBC/PKCS5Padding",
               "stableId" : "openidm-sym-default",
               "salt" : "<hashValue>",
               "data" : "<encryptedValue>",
               "keySize" : 16,
               "purpose" : "idm.config.encryption",
               "iv" : "<encryptedValue>",
               "mac" : "<hashValue>"
           }
       }
    },
    "scope" : [
        "auth"
    ],
...
```

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the admin UI. When you've registered a user with a Wordpress social identity, you can verify this by selecting **Manage > Wordpress**, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: WordPress Social Identity Provider Configuration Details.

Configure user registration to link to WordPress

Once you've configured the WordPress social identity provider, you can activate it through User Registration. To do so in the admin UI, select **Configure > User Registration**, and activate that feature. Under the **Social** tab that appears, enable **Social Registration**. For more information on IDM user self-service features, refer to Self-Service End User UI.

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

WordPress social identity provider configuration details

You can set up the WordPress social identity provider through the admin UI or in a `conf/identityProvider-wordpress.json` file. IDM generates the `identityProvider-wordpress.json` file when you configure and enable this social identity provider in the admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the admin UI WordPress Provider pop-up window, along with associated information in the `identityProvider-wordpress.json` file:

WordPress social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your WordPress App
Client Secret	`clientSecret`	Used with the Client ID to access the applicable WordPress API
Scope	`scope`	An array of strings that allows access to user data; refer to WordPress's OAuth2 Authentication Documentation.
Authorization Endpoint	`authorizationEndpoint`	Typically `https://public-api.wordpress.com/oauth2/authorize`; known as a WordPress *Authorize URL*.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://public-api.wordpress.com/oauth2/token`; known as a WordPress *Request Token URL*.
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; typically `https://public-api.wordpress.com/rest/v1.1/me/`
Not in the admin UI	`name`	Name of the social identity provider
Not in the admin UI	`type`	Authentication module
Not in the admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider
Not in the admin UI	`propertyMap`	Mapping between WordPress and IDM

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

Yahoo social identity provider

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

Set up Yahoo

To set up Yahoo as a social identity provider, navigate to the following page: Yahoo OAuth 2.0 Guide. You'll need a Yahoo account. You can then navigate to the Create an App page, where you can follow the Yahoo process to create a new web application with the following information:

- Application Name
- Web Application

- Callback Domain, such as `openidm.example.com`; required for IDM
- API Permissions; for whatever you select, choose *Read/Write*. IDM only reads Yahoo user information.

When complete and saved, you should see a `Client ID` and `Client Secret` for your new web app.

> **Note**
>
> Yahoo supports URLs using only HTTPS, only on port 443. For more information on how to configure IDM to use these ports, refer to Host and port information.

Configure Yahoo as a social identity provider

To configure a Yahoo social identity provider using the admin UI:

1. From the navigation bar, click **Configure > Social ID Providers**.
2. On the **Social Identity Providers** page, enable **Yahoo**.
3. In the **Yahoo Provider** window, enter applicable values in the fields, and click **Save**. For a complete list of fields, refer to Yahoo Social Identity Provider Configuration Details.

After you save the social identity provider configuration, IDM generates a `conf/identityProvider-yahoo.json` file:

```
{
    "provider" : "yahoo",
    "scope" : [
        "openid",
        "sdpp-w"
    ],
    "uiConfig" : {
        "iconBackground" : "#7B0099",
        "iconClass" : "fa-yahoo",
        "iconFontColor" : "white",
        "buttonClass" : "fa-yahoo",
        "buttonDisplayName" : "Yahoo",
        "buttonCustomStyle" : "background-color: #7B0099; border-color: #7B0099; color:white;",
        "buttonCustomStyleHover" : "background-color: #7B0099; border-color: #7B0099; color:white;"
    },
```

The next part of the file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the admin UI. When you've registered a user with a Yahoo social identity, you can verify this by selecting **Manage > Yahoo**, and then selecting a user.

Next, there's the part of the file that you may have configured through the admin UI, plus additional information on the `redirectUri`, the `configClass`, and the `authenticationIdKey`:

```
"authorizationEndpoint" : "https://api.login.yahoo.com/oauth2/request_auth",
"tokenEndpoint" : "https://api.login.yahoo.com/oauth2/get_token",
"wellKnownEndpoint" : "https://api.login.yahoo.com/.well-known/openid-configuration",
"issuer" : "https://api.login.yahoo.com",
"clientId" : "<Client_ID_Name>",
"clientSecret" : {encrypted-client-secret},
"authenticationIdKey" : "sub",
"redirectUri" : "https://openidm.example.com/",
"basicAuth" : false,
"configClass" : "org.forgerock.oauth.clients.oidc.OpenIDConnectClientConfiguration",
"enabled" : true
```

If you need more information about the properties in this file, refer to the following appendix: Yahoo Social Identity Provider Configuration Details.

Configure user registration to link to Yahoo

Once you've configured the Yahoo social identity provider, you can activate it through User Registration. To do so in the admin UI, select **Configure > User Registration**, and activate that feature. Under the **Social** tab that appears, enable **Social Registration**. For more information on IDM user self-service features, refer to Self-service end user UI.

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Yahoo social identity provider configuration details

You can set up the Yahoo social identity provider through the admin UI or in a `conf/identityProvider-yahoo.json` file. IDM generates the `identityProvider-yahoo.json` file when you configure and enable this social identity provider in the admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the admin UI Yahoo Provider pop-up window, along with associated information in the `identityProvider-yahoo.json` file.

> **Note**
>
> Yahoo supports URLs using only HTTPS, only on port 443. For more information on how to configure IDM to use these ports, refer to Host and port information.

Yahoo social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your Yahoo App.
Client Secret	`clientSecret`	Used with the Client ID to access the applicable Yahoo API.
Scope	`scope`	An array of strings that allows access to user data.
Authorization Endpoint	`authorizationEndpoint`	Typically, `https://api.login.yahoo.com/oauth2/request_auth`; known as a Yahoo **Authorize URL**.

Property (UI)	Property (JSON file)	Description
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token. Typically, `https://api.login.yahoo.com/oauth2/get_token`.
Well-Known Endpoint	`wellKnownEndpoint`	Access for other URIs. Typically, `https://login.yahoo.com/.well-known/openid-configuration`.
Issuer	`issuer`	The token issuer. Typically, `https://api.login.yahoo.com`.
Not in the admin UI	`provider`	Name of the social identity provider.
Not in the admin UI	`configClass`	Configuration class for the authentication module.
Not in the admin UI	`basicAuth`	Whether to use basic authentication.
Not in the admin UI	`propertyMap`	Mapping between Yahoo and IDM.

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

Custom social identity provider

Important

Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

As suggested in the introduction to this chapter, you'll need to take four basic steps to configure a custom social identity provider:

- Prepare IDM
- Set Up a Custom Social Identity Provider
- Configure a Custom Social Identity Provider
- Configure User Registration to Link to a Custom Provider
- Custom Social Identity Provider Configuration Details

Note

These instructions require the social identity provider to be *fully* compliant with The OAuth 2.0 Authorization Framework or the OpenID Connect standards.

Prepare IDM

While IDM includes provisions to work with OpenID Connect 1.0 and OAuth 2.0 social identity providers, connections to those providers are not supported, other than those specifically listed in this chapter. If you haven't already, copy `/path/to/openidm/samples/example-configurations/self-service/identityProviders.json` to your project's `conf/` directory.

To set up another social provider, first add a code block to `conf/identityProviders.json`:

```
{
    "provider" : "<providerName>",
    "authorizationEndpoint" : "",
    "tokenEndpoint" : "",
    "userInfoEndpoint" : "",
    "wellKnownEndpoint" : "",
    "clientId" : "",
    "clientSecret" : "",
    "uiConfig" : {
         "iconBackground" : "",
         "iconClass" : "",
         "iconFontColor" : "",
         "buttonImage" : "",
         "buttonClass" : "",
         "buttonCustomStyle" : "",
         "buttonCustomStyleHover" : "",
         "buttonDisplayName" : ""
    },
    "scope" : [ ],
    "authenticationIdKey" : "",
    "schema" : {
       "id" : "urn:jsonschema:org:forgerock:openidm:identityProviders:api:<providerName>",
       "viewable" : true,
       "type" : "object",
       "$schema" : "http://json-schema.org/draft-03/schema",
       "properties" : {
          "id" : {
             "title" : "ID",
             "viewable" : true,
             "type" : "string",
             "searchable" : true
          },
          "name" : {
             "title" : "Name",
             "viewable" : true,
             "type" : "string",
             "searchable" : true
          },
          "first_name" : {
             "title" : "First Name",
             "viewable" : true,
             "type" : "string",
             "searchable" : true
          },
          "last_name" : {
             "title" : "Last Name",
             "viewable" : true,
             "type" : "string",
             "searchable" : true
          },
          "email" : {
             "title" : "Email Address",
             "viewable" : true,
             "type" : "string",
             "searchable" : true
          },
          "locale" : {
             "title" : "Locale Code",
             "viewable" : true,
             "type" : "string",
```

```
                "searchable" : true
             }
          },
          "order" : [
             "id",
             "name",
             "first_name",
             "last_name",
             "email",
             "locale"
          ],
          "required" : [ ]
       },
       "propertyMap" : [
          {
             "source" : "id",
             "target" : "id"
          },
          {
             "source" : "name",
             "target" : "displayName"
          },
          {
             "source" : "first_name",
             "target" : "givenName"
          },
          {
             "source" : "last_name",
             "target" : "familyName"
          },
          {
             "source" : "email",
             "target" : "email"
          },
          {
             "source" : "email",
             "target" : "username"
          },
          {
             "source" : "locale",
             "target" : "locale"
          }
       ],
       "redirectUri" : "http://openidm.example.com:8080/",
       "configClass" : "org.forgerock.oauth.clients.oidc.OpenIDConnectClientConfiguration",
       "basicAuth" : false,
       "enabled" : true
    },
```

Modify this code block for your selected social provider. Some of these properties may appear under other names. For example, some providers specify an `App ID` that you'd include as a `clientId`.

Additional changes may be required, especially depending on how the provider implements the OAuth2 or OpenID Connect standards.

In the `propertyMap` code block, you should substitute the properties from the selected social identity provider for various values of `source`. Make sure to trace the property mapping through `selfservice.propertymap.json` to the Managed User property shown in `managed.json`. For more information on this multi-step mapping, refer to Many social identity providers, one schema.

As shown in OpenID connect authorization code flow, user provisioning information goes through the User Info Endpoint. Some providers, such as LinkedIn and Facebook, may require a list of properties with the endpoint. Consult the documentation for your provider for details.

For more information on the `uiConfig` code block, refer to Social identity provider button and badge properties.

Both files, `identityProviders.json` and `identityProvider-custom.json`, should include the same information for the new `custom` identity provider. For property details, refer to Custom Social Identity Provider Configuration Details.

Once you've included information from your selected social identity provider, proceed with the configuration process. You'll use the same basic steps described for other specified social providers.

Set up a custom social identity provider

Every social identity provider should be able to provide the information you need to specify properties in the code block shown in Prepare IDM.

In general, you'll need an `authorizationEndpoint`, a `tokenEndpoint` and a `userInfoEndpoint`. To link to the custom provider, you'll also have to copy the `clientId` and `clientSecret` that you created with that provider. In some cases, you'll get this information in a slightly different format, such as an `App ID` and `App Secret`.

For the `propertyMap`, check the `source` properties. You may need to revise these properties to match those available from your custom provider.

For examples, refer to the specific social identity providers documented in this chapter.

Configure a custom social identity provider

1. To configure a custom social identity provider, log in to the admin UI and navigate to **Configure > Social ID Providers**.
2. Enable the custom social identity provider. The name you refer to is based on the `name` property in the relevant code block in the `identityProviders.json` file.
3. If you haven't already done so, include the values provided by your social identity provider for the properties shown. For more information, refer to the following appendix: Custom Social Identity Provider Configuration Details.

Configure user registration to link to a custom provider

Once you've configured a custom social identity provider, you can activate it through User Registration. To do so in the admin UI, select **Configure > User Registration**, and under the **Social** tab, enable the option associated with **Social Registration**. For more information about user self-service features, refer to IDM user interface.

When you enable social identity providers, you're allowing users to register on IDM through all active social identity providers.

Custom social identity provider configuration details

When you set up a custom social identity provider, starting with Prepare IDM, you'll refer to configuration details in your `conf/identityProviders.json` file. The following table includes the information shown in the relevant admin UI pop-up window.

IDM generates the content of `identityProvider-custom.json` after you configure and enable the custom social identity provider using the admin UI. Before you can activate this feature in the admin UI, copy `/path/to/openidm/samples/example-configurations/self-service/identityProviders.json` to your project's `conf/` directory. You can also manually create this file.

Custom social identity provider configuration properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your social identity provider
Client Secret	`clientSecret`	Used with the Client ID
Scope	`scope`	An array of strings that allows access to user data; varies by provider.
Authorization Endpoint	`authorizationEndpoint`	Every social identity provider should have an authorization endpoint to authenticate end users.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token.
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields.
Not in the admin UI	`name`	Name of the social identity provider
Not in the admin UI	`type`	Authentication module
Not in the admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider
Not in the admin UI	`propertyMap`	Mapping between the social identity provider and IDM

For information on social identity provider buttons and badges, refer to Social identity provider button and badge properties.

Social providers authentication module

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

The `SOCIAL_PROVIDERS` authentication module incorporates the requirements from social identity providers who rely on either the OAuth2 or OpenID Connect standards. The Social Providers authentication module is disabled by default. To configure or enable this module in the admin UI, select **Configure > Authentication**, choose the **Modules** tab, then select **Social Providers** from the list of modules.

Authentication settings can be configured from the admin UI.

The authentication properties are described in detail in Authentication and session module configuration.

Account claiming: links between accounts and social identity providers

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

If your users have one or more social identity providers, they can link them to the same IDM user account. This section assumes that you have configured one or more of the social identity providers described in Social registration.

Conversely, you should not be able to link more than one IDM account with a single social identity provider account.

When social accounts are associated with an IDM account, IDM creates a managed record, which uses the name of the social identity provider name as the managed object type, and the subject is used as the `_id`. This combination has a unique constraint; if you try to associate a second IDM account with the same social account, IDM detects a conflict, which prevents the association.

The default process uses the email address associated with the account. Once you've configured social identity providers, you can see this filter in the `selfservice-socialUserClaim.json` file:

```
{
    "name" : "socialUserClaim",
    "identityServiceUrl" : "managed/user",
    "claimQueryFilter" : "/mail eq \"{{mail}}\""
},
```

You can modify the `claimQueryFilter` to a different property such as `telephoneNumber`. Make sure that property is:

- Set to "required" in the `managed.json` file; the default list for managed users is shown here:

  ```
  "required" : [
      "userName",
      "givenName",
      "sn",
      "mail"
  ]
  ```

- Unique; for example, if multiple users have the same telephone number, IDM responds with error messages shown in When Multiple Users have the Same Email Address.

Based on the `claimQueryFilter`, what IDM does depends on the following scenarios:

- When the Email Address is New
- When One User has the Same Email Address
- When Multiple Users have the Same Email Address

When the email address is new

When you register with a social identity provider, IDM checks the email address of that account against the managed user data store.

If that email address doesn't exist for any IDM managed user, IDM takes available identifying information, and pre-populates the self-registration screen. If all required information is included, IDM proceeds to other screens, depending on what you've activated in this section: Additional configuration.

When one user has the same email address

When you register with a social identity provider, IDM checks the email address of that account against the managed user data store.

If that email address exists for *one* IDM managed user, IDM gives you a chance to link to that account, with the following message:

```
We found an existing account with the same email address <substitute email address>. To continue, please
enter your password to link accounts.
```

In the text box, users are expected to enter their IDM account password.

When multiple users have the same email address

When you register with a social identity provider, IDM checks the email address of that account against the managed user data store.

If that email address exists for *multiple* IDM managed users, IDM denies the login attempt, with the following error message:

```
Unable to authenticate using login provider
```

IDM denies further attempts to login with that account with the following message:

```
Forbidden request error
```

For information about customizing the End User UI, refer to the Github repository: ForgeRock/end-user-ui.

The process for end users

When your users register with a social identity provider, as defined in Social registration, they create an account in the IDM managed user data store. As an end user, you can link additional social identity providers to that data store, from the End User UI:

1. Navigate to the End User UI. For example, `http://idm.example.com:8080`.
2. Log in to the account, either as an IDM user, or with a social identity provider.
3. Navigate to **Profile** > **Social Sign-in**.

   The list of configured social identity providers displays.

4. Connect to the social identity providers of your choice. Unless you've already signed in with that social provider, you are prompted to log in to that provider.

5. To test the result, log out and log back in, using the link for the newly linked social identity provider.

Reviewing linked accounts as an administrator

You can review social identity accounts linked to an IDM account, from the admin UI and from the command line. You can disable or delete social identity provider information for a specific user from the command line, as described in Reviewing Linked Accounts Over REST.

When you activate a social identity provider, IDM creates a new managed object for that provider. You can review that managed object in the `managed.json` file, as well as in the admin UI, by selecting **Configure > Managed Objects**.

The information shown is reflected in the schema in the `identityProvider-providername.json` file for the selected provider.

> **Note**
>
> Do not edit social identity provider profile information in IDM. Any changes won't be synchronized with the provider.

Reviewing linked accounts over REST

To identify linked social identity provider accounts for a user, you must specifically add the `idps` field to your user query. For example, the following query shows bjensen's linked social identity information:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=userName+eq+'bjensen'&_fields=idps"
{
  "result": [
    {
      "_id": "bjensen",
      "_rev": "000000003062291c",
      "idps": [
        {
          "_ref": "managed/google/108246554379618660085",
          "_refResourceCollection": "managed/google",
          "_refResourceId": "108246554379618660085",
          "_refProperties": {
            "_id": "ba01a4c3-8a7f-468b-8b09-95f5d34f05ea",
            "_rev": "0000000098619792"
          }
        }
      ]
    }
  ],
  ...
}
```

For more information about a specific social identity provider, query the identity *relationship* using the referred resource ID. The following example shows the information collected from the Google provider for bjensen:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/google/108246554379618660085"
{
  "_id": "108246554379618660085",
  "_rev": "00000000e5cace4d",
  "sub": "108246554379618660085",
  "name": "Barbara Jensen",
  "given_name": "Barbara",
  "family_name": "Jensen",
  "picture": "https://lh3.googleusercontent.com/-XdUIqdMkCWA/AAAAAAAAAAI/AAAAAAAAAAA/4252rscbv5M/
photo.jpg",
  "email": "babs.jensen@gmail.com",
  "email_verified": true,
  "locale": "en",
  "_meta": {
    "subject": "108246554379618660085",
    "scope": [
      "openid",
      "profile",
      "email"
    ],
    "dateCollected": "2018-03-08T02:07:27.882"
  }
}
```

When a user disables logins through one specific social identity provider in the End User UI, that sets `"enabled" : false` in the data for that provider. However, that user's social identity information is preserved.

Alternatively, you can use a REST call to disable logins to a specific social identity provider. The following REST call removes a user's ability to log in through Google:

```
curl  \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-type: application/json" \
--request POST \
"http://localhost:8080/openidm/managed/user/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb?
_action=unbind&provider=google"
```

In this case, the REST call deletes all Google social identity provider information for that user.

Reviewing linked accounts from the admin UI

When you configure a social identity provider, IDM includes two features in the admin UI.

- The ability to review the social identity accounts linked to specific users. To refer to how this works, log in to the admin UI, and select **Manage > User**, and select a user. Under the **Identity Providers** tab, you can review the social identity providers associated with a specific account.
- A managed object for each provider. For example, if you've enabled Google as a social identity provider, select **Manage > Google**. On the **Google List** page, you can select the ID for any Google social identity account that has been used or linked to an existing IDM account, and review the profile information shared from the provider.

Social identity providers over REST

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

You can identify the current status of configured social identity providers with the following REST call:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/authentication'
```

The output that you refer to includes JSON information from each configured social identity provider, as described in the `identityProvider-provider` file in your project's `conf/` subdirectory.

One key line from this output specifies whether the social identity provider is enabled:

```
"enabled" : true
```

If the `SOCIAL_PROVIDERS` authentication module is disabled, you'll refer to the following output from that REST call:

```
{
   "providers" : [ ]
}
```

For more information, refer to Social providers authentication module.

If the `SOCIAL_PROVIDERS` module is disabled, you can still review the standard configuration of each social provider (enabled or not) by running the same REST call on a different endpoint (do not forget the `s` at the end of `identityProviders`):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/identityProviders'
```

Note

If you have not configured a social identity provider, you'll refer to the following output from the REST call on the `openidm/identityProviders` endpoint:

```
{
"providers" : [ ]
}
```

You can still get information about the available configuration for social identity providers on a slightly different endpoint:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/config/identityProviders'
```

The `config` in the endpoint refers to the configuration, starting with the `identityProviders.json` configuration file. Note how it matches the corresponding term in the endpoint.

You can review information for a specific provider by including the name with the endpoint. For example, if you've configured LinkedIn as described in LinkedIn social identity provider, run the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
'http://localhost:8080/openidm/config/identityProvider/linkedIn'
```

The above command differs in subtle ways. The `config` in the endpoint points to configuration data. The `identityProvider` at the end of the endpoint is singular, which matches the corresponding configuration file, `identityProvider-linkedIn.json`. And `linkedIn` includes a capital `I` in the middle of the word.

In a similar fashion, you can delete a specific provider:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
'http://localhost:8080/openidm/config/identityProvider/linkedIn'
```

If you have the information needed to set up a provider, such as the output from the previous two REST calls, you can use the following command to add a provider:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
 --header "Accept-API-Version: resource=1.0" \
 --header "Content-type: application/json" \
 --request PUT \
--data '{
 <Include content from an identityProvider-linkedIn.json file>
}' \
'http://localhost:8080/openidm/config/identityProvider/linkedIn'
```

IDM incorporates the given information in a file named for the provider, in this case, `identityProvider-linkedIn.json`.

You can even disable a social identity provider with a `PATCH` REST call, as shown:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-type: application/json" \
--request PATCH \
--data '[
   {
      "operation":"replace",
      "field" : "enabled",
      "value" : false
   }
]' \
'http://localhost:8080/openidm/config/identityProvider/linkedIn'
```

You can reverse the process by substituting `true` for `false` in the previous `PATCH` REST call.

You can manage the social identity providers associated with individual users over REST, as described in Social identity providers over REST.

Test social identity providers

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

Once social identity provider configuration is complete, you should test the provider:

1. Navigate to the login screen for the End User UI, `https://openidm.example.com:8443`.
2. Click **Register** on the login page.
3. Click the applicable link to sign in with the selected social identity provider.

   > **Note**
   >
   > If you do not refer to a link to sign in with any social identity provider, make sure that Social Registration is enabled. In the admin UI, select **Configure > User Registration**.

   > **Warning**
   >
   > If you refer to a redirect URI error from a social identity provider, check the configuration for your web application in the social identity provider developer console. There may be a mistake in the redirect URI or redirect URL.

4. Follow the prompts from your social identity provider to log in to your account.

   > **Note**
   >
   > If there is a problem with the interface to the social identity provider, you might refer to a **Register Your Account** screen with information acquired from that provider.

5. Because security questions are enabled by default, you must add at least one security question and answer to proceed. For more information, refer to Security questions.

   When the Social ID registration process is complete, you are redirected to the End User UI at `https://openidm.example.com:8443`.

6. You should now be able to use the sign in link for your social identity provider.

Social registration scenarios

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

When users connect to IDM with a social identity provider, it could be the first time they're connecting to your system. They could already have an regular IDM account. They could already have registered with a different social identity provider. This section describes what happens during the self-registration process. The process varies depending on whether there's an existing account in the IDM managed user store.

The flow varies slightly if the user already exists in IDM.

Figure 1. The flow varies slightly if the user already exists in IDM.

The following list describes each item in the flow shown in the adjacent figure:

1. From the IDM End User UI, the user selects the `Register` link
2. The self-registration Interface returns a `Register Your Account` page at `{hostname}/#/registration` with a list of configured providers.
3. The user then selects one configured social identity provider.
4. IDM connects to the selected social identity provider.
5. The social identity provider requests end user authentication.
6. The end user authenticates with the social identity provider.
7. The social identity provider prompts the user to accept sharing selected account information.
8. The user accepts the conditions presented by the social identity provider.
9. The social identity provider notifies IDM of the user registration request.
10. IDM passes responsibility to the administrative interface.
11. IDM uses the email address from the social identity provider, and compares it with email addresses of existing managed users.
12. If the email address is found, IDM links the social identity information to that account (and skips to step 16).
13. IDM returns to the self-registration (Self-Service) interface.
14. The self-registration interface prompts the user for additional information, such as security questions, and reCAPTCHA, if configured per Google reCAPTCHA.
15. The user responds appropriately.
16. IDM creates a new managed user. If the user has already been created, IDM reviews data from the social identity provider, and updates the user data for the managed/*provider* to conform. In this case, the *provider* is a social identity provider such as Google.
17. The user is redirected to the `Success URL`.

Social identity widgets

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

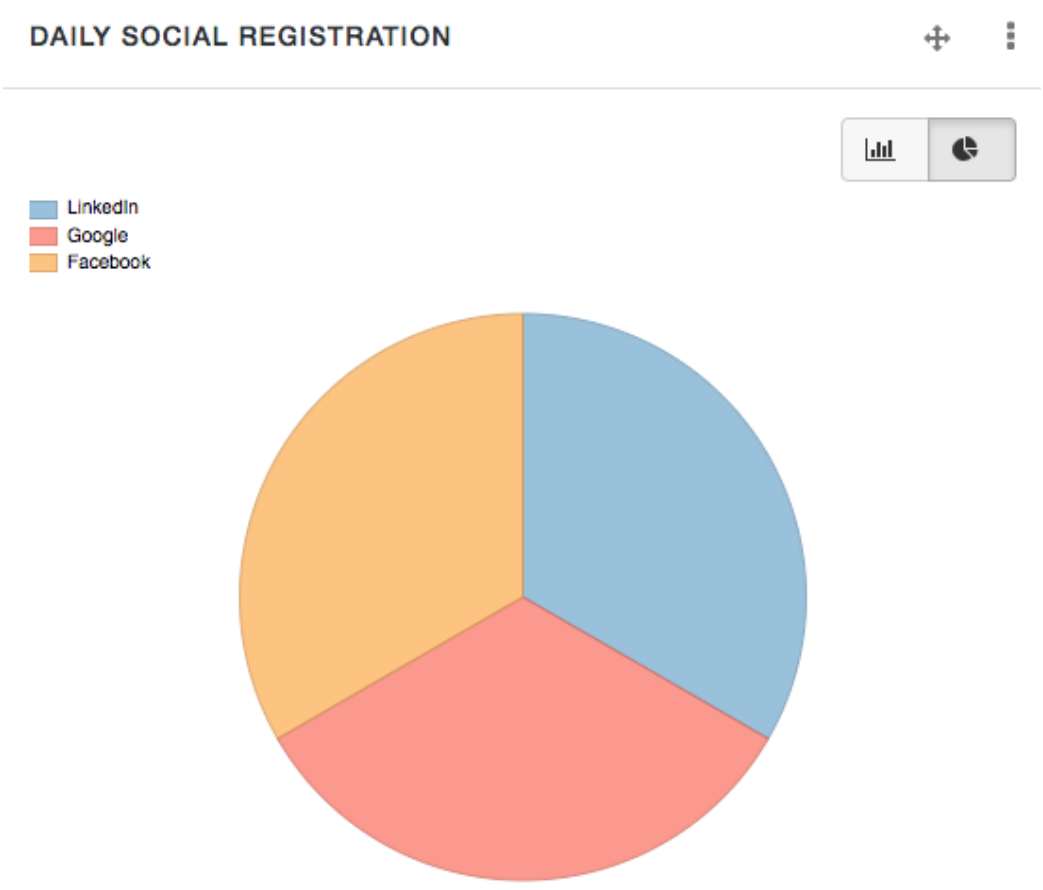
The admin UI includes widgets that can help you measure the success of your social identity efforts. To add these widgets, take the following steps:

1. Log in to the admin UI.
2. From the navigation bar, click **Dashboards**, and select a dashboard.

   For more information about managing dashboards in the UI, refer to Manage dashboards.
3. Click **Add Widget**.
4. In the **Add Widget** window, scroll down to the **Social** item, and select one of the following graphical widgets:
   - Social Registration (year)
   - Daily Social Registration
   - Daily Social Logins

   A preview of the widget displays. Your IDM system must contain some social data to display the preview correctly.
5. Click **Settings** to configure the widget.

The following example shows daily social registrations:



Social identity provider button and badge properties

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

You can configure buttons and badges for each social identity provider, using the admin UI or by editing the associated `identityProvider-name.json` file. The admin UI displays examples during social identity provider configuration.

Badges appear in the admin UI under **Configure > Social ID Providers** , and in the End User UI under **My Account > Sign-in & Security > Social Sign-in**.

Buttons appear in the IDM login screens, and when you select **Register** from the End User UI login screen.

Example Button	Example Badges
	

How IDM displays buttons and badges changes based on how many social identity providers are enabled:

- For up to three social identity providers, IDM displays large buttons, with the text **Register with Provider**.
- For four or more social identity providers, IDM displays smaller buttons with icons.

> **Note**
>
> For seven or more social identity providers, horizontal scrolling may be required.

Properties for Social Identity Provider Buttons and Badges

Property (UI)	Property (JSON file)	Description
Badge background color	`iconBackground`	Color for the social identity provider icon.
Badge icon classname	`iconClass`	Name of the icon class. Can be a Font Awesome name like `fa-google`.
Badge font color	`iconFontColor`	Color for the social identity provider icon font.
Button image path	`buttonImage`	Looks in `openidm/ui/admin/extension` and then `openidm/ui/admin/default` for an image file. Takes precedence over the *Button icon classname*.
Button icon classname	`buttonClass`	Name for the social identity provider class. Can be a Font Awesome name like `fa-yahoo`.

Property (UI)	Property (JSON file)	Description
Button display name	`buttonDisplayName`	Name to display on large buttons.
Button styles	`buttonCustomStyle`	Custom styles, such as `background-color: #7B0099; border-color: #7B0099; color:white;`.
Button hover styles	`buttonCustomStyleHover`	Custom styles for the hover state of a button, such as `background-color: #7B0099; border-color: #7B0099; color: white;`.

Progressive profile

Progressive profile completion lets you gather profile attributes asynchronously to enrich your users' profile data, and enhance engagement with your customer base. Profile completion requires the creation of one or more *forms* to collect user data.

IDM implements progressive profile completion as a default self-service process. You can use this process as an example of how to build additional functionality into a custom client application, using the Self-Service REST API.

After activating Self-registration, users need only the following information to register:

- User name
- First name
- Last name
- Email address

Progressive profile completion lets you collect additional information, limited by the attributes defined in the `managed.json` file for your project.

In the following sections, you'll examine how you use progressive profile completion to ask or require more information from users. You're limited only by what properties are defined in your project's `managed.json` file.

Progressive profile completion form

If you're testing progressive profile completion, you can start from the `selfservice-profile.json` file in the following directory: `openidm/samples/example-configurations/self-service/`

Copy this file to your project's `conf/` directory and start IDM. After the conditions shown in this configuration file are met, end users will refer to a form prompting them to add a telephone number.

```
{
    "stageConfigs" : [
        {
            "name" : "conditionaluser",
            "identityServiceUrl" : "managed/user",
            "condition" : {
                "type" : "loginCount",
                "interval" : "at",
                "amount" : 25
            },
            "evaluateConditionOnField" : "user",
            "onConditionTrue" : {
                "name" : "attributecollection",
                "identityServiceUrl" : "managed/user",
                "uiConfig" : {
                    "displayName" : "Add your telephone number",
                    "purpose" : "Help us verify your identity",
                    "buttonText" : "Save"
                },
                "attributes" : [
                    {
                        "name" : "telephoneNumber",
                        "isRequired" : true
                    }
                ]
            }
        }
    ]
}
```

The following table includes a detailed list of each property shown in this file:

The `selfservice-profile.json` *File*

Property	Description
`stageConfigs`	Progressive profile completion is a stage of user self-service.
`name`	`conditionaluser` sets up conditions for end users.
`identityServiceUrl`	`managed/user` specifies IDM Managed Users.
`condition`	Condition when to display the form.
`type`	Type of `condition`; for a list of conditions, refer to Progressive Profile Completion Conditions.
`evaluateConditionOnField`	IDM evaluates the condition, per `user`.
`onConditionTrue`	Presents the form with the following properties.
`name`	Data that you collect with the form is an `attributeCollection`.

Property	Description
`uiConfig`	Labels to include the in the form seen by the end user.
`displayName`	Form title.
`purpose`	Form explanation.
`buttonText`	Customizable.
`attributes`	Attribute name from `managed.json`.
`isRequired`	If an end user has to enter data to complete a connection to IDM.

The default progressive profile completion process involves two mandatory stages:

- Conditional User Stage
- Attribute collection stage

With the previous configuration, users logging in to the End User UI must submit a telephone number on the 25th login.

Progressive profile completion conditions

You can set up a number of different conditions for when users are prompted to add information to their profiles. IDM includes the following pre-defined criteria:

`loginCount`

May specify `at` or `every` number of logins, as defined by the following value: `amount`.

> **Note**
>
> End users can bypass progressive profile completion screens, when configured with a `loginCount`. Every time they refer to such a request, they can open a new browser window to bypass that request, and log in to the End User UI. They won't have to provide the information requested, even if you've set the attribute as Required under the Attributes tab.

`timeSince`

May specify a time since the user was created, the `createDate`, in `years`, `months`, `weeks`, `days`, `hours`, and `minutes`.

`profileCompleteness`

Based on the number of items completed by the user from `managed.json`, in percent, as defined by `percentLessThan`; for more information, refer to Defining Overall Profile Completion.

`propertyValue`

Based on the value of a specific user entry, such as `postalAddress`, which can be defined by Presence Expressions.

Custom progressive profile conditions

You can also set up custom conditions with query filters and scripts. These criteria may deviate from standard query filters described in Construct Queries and standard scripted conditions described in Add Conditional Policy Definitions.

- A queryFilter. For example, the following query filter checks user information for users who live in the city of Portland:

```
"condition" : {
       "type" : "queryFilter",
       "filter" : "/city eq \"Portland\""
       },
```

  In addition, you can also reference metadata, as described in Track User Metadata. For example, the following query filter searches for users with:

  - A `loginCount` greater than or equal to five.
  - Does not have a telephone number:

```
"filter" : "(/_meta/loginCount ge 5 and !(/telephoneNumber pr))"
```

> **Warning**
>
> If you include `_meta` in query filters, the admin UI will not work for the subject progressive profiling form.
> While it's technically possible to include a number like `5` in the admin UI with the query filter, IDM would write the number as a string to the `selfservice-profile.json` file. You'd still have to change that number directly in the noted file.

- An inline script (`scripted`), or a reference to a script file; IDM works with scripts written in either JavaScript or Groovy. For example, you could set up a script here:

```
"condition" : {
       "type" : "scripted",
       "script" : {
       "type" : "text/javascript",
       "globals" : { },
       "source" : "<some script code>"
       },
```

  Alternatively, you could point to some JavaScript or Groovy file:

```
"condition" : {
       "type" : "scripted",
       "script" : {
       "type" : "text/javascript",
       "globals" : { },
       "file" : "path/to/someScript.js"
       },
```

For the script code, you'll need to reference fields directly, and not by `object.field`. For example, the following code would test for the presence of a telephone number:

```
typeof telephoneNumber === 'undefined' || telephoneNumber === ''
```

While you can also reference metadata for scripts, you can't check for all available fields, as there is no outer `object` field. However, you can see fields that are part of the user object.

Configuring progressive profile completion through the admin UI

The UI is straightforward; in the admin UI, when you select Configure > Progressive Profile, you'll add a *New Form*, with:

- Attributes defined in `managed.json`.
- Conditions that may be based on a query filter, a script, or pre-defined criteria such as number of logins.

What you configure in the admin UI is written to the `selfservice-profile.json` file. The information under the following admin UI Progressive Profile Completion page tabs is written to the following code blocks in that file:

- Details: `uiConfig`
- Display Condition: `condition`
- Attributes: `attributes`

Warning

When you use the UI, you *must* specify a property under the Attributes tab. Otherwise, IDM won't display a Progressive Profile form. To specify a property, select Configure > Progressive Profile. Select a Progressive Profile form > Attributes tab > Add a Property. Be sure to select an Attribute Name based on user properties configured in the `managed.json` file.

The `auth.profile.json` file

Note

To use `auth.profile.json`, copy the file from `/path/to/openidm/samples/example-configurations/self-service/` to your project's `conf/` directory.

In some circumstances, you may wish to create a temporary role for users who are in the middle of progressive profile completion, such as if you wish to enable access to an endpoint, while prohibiting access to other parts of the End User UI (as well as the rest of IDM).

To do this, you may optionally define an `authenticationRole` in `auth.profile.json`, which you can use as a role assignment in `access.json` or elsewhere.

For example, if you wished to assign access to a custom endpoint for users who have incomplete profiles, you could modify `auth.profile.json` to include a custom `authenticationRole` called `incomplete-profile`:

```
{
    "profileEnhancementProcesses": [
        "selfservice/termsAndConditions",
        "selfservice/kbaUpdate",
        "selfservice/profile"
    ],
    "authenticationRole": "incomplete-profile",
    "authorizationRole": "internal/role/openidm-authorized"
}
```

You could then give access to this role to your custom endpoint in `access.json`:

```
{
    "pattern"    : "endpoint/extra-steps",
    "roles"      : "incomplete-profile",
    "methods"    : "read",
    ...
},
```

Access for these and other roles is governed by the `access.json` script. For more information, refer to Configure Access Control in access.json.

The role specified in `authenticationRole` can be an existing role, or it can be a placeholder string. If it is a placeholder, it will not function as a real role, but can still be used for access in `access.json`, and will appear in access and authentication log files in the `openidim/audit` directory.

Progressive profile completion and metadata

Progressive profile completion requires that you track object metadata. Configure tracking of the following data:

- `createDate`: The date the user was created; used in the `onCreateUser.js` script in the `openidm/bin/defaults/script` directory.
- `loginCount`: The number of logins, by user.
- `stagesCompleted`: Used to track progressive profile forms, and whether they've been completed, by user.

User acceptance of Terms & Conditions is tracked by default (see Terms & Conditions).

Defining overall profile completion

A user profile is based on every item in `managed.json` where both `viewable` and `userEditable` are set to `true`. Every qualifying item has equal weight.

So, if there are 20 qualifying items in `managed.json`, a user who has entries for 10 items has a *Profile completion percentage* of 50.

Progressive profile REST requests

The following REST requests and responses demonstrate the flow through a profile completion process, given the previous configuration:

1. Client attempts a login for the 25th time:

```
curl \
 --header "X-OpenIDM-Username: bjensen" \
 --header "X-OpenIDM-Password: Passw0rd" \
 --header "X-OpenIDM-NoSession: false" \
 --request POST \
 "https://localhost:8443/openidm/authentication?_action=login"
{
  "_id": "login",
  "authorization": {
    "userRolesProperty": "authzRoles",
    "processesRequired": true,
    "component": "managed/user",
    "authLogin": true,
    "authenticationIdProperty": "username",
    "roles": [],
    "ipAddress": "0:0:0:0:0:0:0:1",
    "protectedAttributeList": ["password"],
    "requiredProfileProcesses": ["selfservice/profile"],
    "id": "51c6c46d-3d7b-4671-8295-0c8ee39e8549",
    "moduleId": "MANAGED_USER",
    "queryId": "credential-query"
  },
  "authenticationId": "bjensen"
}
```

> **Note**
>
> The values of the `requiredProfileProcesses` and `roles` properties in the returned output trigger the remainder of the process. If `requiredProfileProcesses` is present and not empty, there are processes that must be completed. Ultimately, the process must return a full access `role` (such as `internal/role/openidm-authorized`) and continue to the user profile page.

2. Server sends a GET request to the `profile` endpoint and returns `"type": "conditionaluser"` and `"tag": "initial"` to start the profile completion process:

```
curl \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-Password: anonymous" \
 --request GET \
 "https://localhost:8443/openidm/selfservice/profile"
{
        "_id": "1",
        "_rev": "991096945",
        "type": "conditionaluser",
        "tag": "initial",
        "requirements": {
                "$schema": "http://json-schema.org/draft-04/schema#",
                "description": "Attribute Details",
                "type": "object",
                "properties": {},
                "attributes": [{
                        "name": "telephoneNumber",
                        "isRequired": true,
                        "schema": {
                                "type": "string",
                                "title": "Telephone Number",
                                "description": "Telephone Number",
                                "viewable": true,
                                "userEditable": true,
                                "pattern": "^\\+?([0-9\\- \\(\\)])*$",
                                "usageDescription": "",
                                "isPersonal": true
                        },
                        "value": null
                }],
                "uiConfig": {
                        "displayName": "Add your telephone number",
                        "purpose": "Help us verify your identity",
                        "buttonText": "Save"
                }
        }
}
```

3. Client submits requirements, in this case, the required profile field. Server response includes `"tag": "end"` and `"success": true` to signal the end of the profile process:

```
curl \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-Password: anonymous" \
 --request POST \
 --data '{
     "input":{
         "attributes":{
             "telephoneNumber":"555-555-1234"
         }
     }
 }'
 "https://localhost:8443/openidm/selfservice/reset?_action=submitRequirements"
{
        "type": "conditionaluser",
        "tag": "end",
        "status": {
                "success": true
        },
        "additions": {}
}
```

Viewing profile completeness

You can view how complete a profile is, presented as the percentage of user-editable attributes that have been filled out on a profile. To do so, send a REST call to the `selfservice/profile/completeness` endpoint:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
 --request GET \
"http://localhost:8080/openidm/selfservice/profile/completeness/managed/user/3a8cabef-
d4a3-4f60-926a-52f27257bde6"
{
  "_id": "managed/user/3a8cabef-d4a3-4f60-926a-52f27257bde6",
  "_rev": "00000000c38d9344",
  "completeness": 42.857143
}
```

Password reset

IDM supports self-service user password reset. When enabled, users who forget their passwords can log in to the IDM End User UI, and can verify their identities with options such as email validation and security questions.

You can also generate random passwords when users are created. For more information, refer to Generating Random Passwords.

Password reset lets registered users reset their own passwords. The following stages can be included in a password reset process:

- Captcha stage (optional)

- User query stage (mandatory)
- Email validation stage (optional)
- KBA security answer verification stage (optional)
- Password reset stage (mandatory)

If all of these stages are configured, the password reset configuration (in `conf/selfservice-profile.json` looks similar to the following:

```
{
    "stageConfigs" : [
        {
            "name" : "captcha",
            "recaptchaSiteKey" : "...",
            "recaptchaSecretKey" : "...",
            "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
        },
        {
            "name" : "userQuery",
            "validQueryFields" : [
                "userName",
                "mail",
                "givenName",
                "sn"
            ],
            "identityIdField" : "_id",
            "identityEmailField" : "mail",
            "identityUsernameField" : "userName",
            "identityServiceUrl" : "managed/user"
        },
        {
            "name" : "emailValidation",
            "identityEmailField" : "mail",
            "emailServiceUrl" : "external/email",
            "emailServiceParameters" : {
                "waitForCompletion" : false
            },
            "from" : "info@example.com",
            "subject" : "Reset password email",
            "mimeType" : "text/html",
            "subjectTranslations" : {
                "en" : "Reset your password",
                "fr" : "R&#233;initialisez votre mot de passe"
            },
            "messageTranslations" : {
                "en" : "...Click to reset your password...",
                "fr" : "...Cliquez pour r&#233;initialiser votre mot de passe..."
            },
            "verificationLinkToken" : "%link%",
            "verificationLink" : "https://localhost:8443/#/passwordreset/"
        },
        {
            "name" : "kbaSecurityAnswerVerificationStage",
            "kbaPropertyName" : "kbaInfo",
            "identityServiceUrl" : "managed/user",
            "kbaConfig" : null
        },
        {
            "name" : "resetStage",
            "identityServiceUrl" : "managed/user",
            "identityPasswordField" : "password"
        }
    ],
    "snapshotToken" : {
        "type" : "jwt",
        "jweAlgorithm" : "RSAES_PKCS1_V1_5",
        "encryptionMethod" : "A128CBC_HS256",
```

```
        "jwsAlgorithm" : "HS256",
        "tokenExpiry" : "300"
    },
    "storage" : "stateless"
}
```

User password reset configuration files

To set up basic user password reset features, you'll need at least the following configuration files:

- `selfservice-reset.json`

  You can find a template version of this file in the following directory: `openidm/samples/example-configurations/self-service`.

- `ui-configuration.json`

  You can find this file in the default IDM project configuration directory, `openidm/conf`.

To set up self-service user password reset registration, enable the following boolean in `ui-configuration.json`:

```
"passwordReset" : true,
```

You can include several features with user password reset, as shown in the following excerpts of the `selfservice-reset.json` file:

- If you've activated Google reCAPTCHA for user self-service registration, you'll refer to the following code block:

  ```
  {
      "name" : "captcha",
      "recaptchaSiteKey" : "<siteKey>",
      "recaptchaSecretKey" : "<secretKey>",
      "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
  },
  ```

  As suggested by the code, you'd substitute the actual `siteKey` and `secretKey` assigned by Google for your domain. For more information, refer to Google reCAPTCHA.

- For password reset, IDM needs to verify user identities. To ensure that password reset links are sent to the right user, include the following code block:

```
{
    "name" : "userQuery",
    "validQueryFields" : [
        "userName",
        "mail",
        "givenName",
        "sn"
    ],
    "identityIdField" : "_id",
    "identityEmailField" : "mail",
    "identityUsernameField" : "userName",
    "identityServiceUrl" : "managed/user"
},
```

This code lets IDM verify user identities by their username, email address, first name (`givenName`), or last name (`sn` , short for surname).

- If you have included email verification, you must configure an outgoing email server. For details about the required addition to `selfservice-registration.json` , refer to Email for password reset.
- If you've configured security questions, users who self-register will have to create questions and answers during the self-registration process.

  If the feature is enabled, users who've been reconciled from external data stores will also be prompted, once, upon their next login, to add security questions and answers. The relevant code block is shown here, which points IDM to other configuration files as discussed in links from this section.

```
{
    "name" : "kbaSecurityAnswerDefinitionStage",
    "kbaConfig" : null
},
```

Configuring password reset from the admin UI

To configure Password Reset from the admin UI, select **Configure > Password Reset**. When you activate **Enable Password Reset**, you'll refer to a `Configure Password Reset Form` that lets you specify the:

- **Identity Resource**, typically `managed/user`
- **Advanced Options, Snapshot Token**, typically a JSON Web Token (JWT)
- **Advanced Options, Token Lifetime**, with a default of 300 seconds

You can also add these settings to the following configuration file: `selfservice-reset.json` . When you modify these settings in the admin UI, IDM creates the file for you.

Email for password reset

To configure emails for password reset, you can add the following code block to the `selfservice-reset.json` file:

```
{
    "name" : "emailValidation",
    "identityEmailField" : "mail",
    "emailServiceUrl" : "external/email",
    "emailServiceParameters" : {
        "waitForCompletion" : false
    },
    "from" : "info@example.com",
    "subject" : "Reset password email",
    "mimeType" : "text/html",
    "subjectTranslations" : {
        "en" : "Reset your password",
        "fr" : "Réinitialisez votre mot de passe"
    },
    "messageTranslations" : {
        "en" : "<h3>Click to reset your password</h3><h4><a href=\"%link%\">Password reset link</a></h4>",
        "fr" : "<h3>Cliquez pour réinitialiser votre mot de passe</h3><h4><a href=\"%link%\">Mot de passe lien de
réinitialisation</a></h4>"
    },
    "verificationLinkToken" : "%link%",
    "verificationLink" : "https://localhost:8443/#/passwordreset/"
},
```

As suggested by the code block, it includes default password reset email messages in English (`en`) and French (`fr`). The `verificationLink` sent with the email takes users to the IDM password reset URL.

As noted in Password reset REST requests, you can make these changes over the following endpoint URI: `/openidm/config/selfservice/reset`

If desired, you can also configure self-service password reset emails through the admin UI. Select **Configure > Password Reset**. If needed, activate the **Enable Password Reset** option, and in the **Email Validation** box, click the ✎ button. The **Configure Validation Email** window displays.

When you use the admin UI to customize password reset emails, you can review the changes in the `selfservice-reset.json` file.

Password reset REST requests

The following REST requests and responses demonstrate the flow through a simple password reset process. To keep the process simple, this flow does not include the Google ReCAPTCHA stage, or the Security Answer Verification stage:

1. Client initiates the password reset,

   The server returns the `initial` tag:

```
curl \
--request GET \
"https://localhost:8443/openidm/selfservice/reset"
{
  "type": "parameters",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Parameters",
    "type": "object",
    "properties": {
      "returnParams": {
        "description": "Parameter named 'returnParams'",
        "type": "string"
      }
    }
  }
}
```

2. Initial requirements submission with an empty payload.

   The server returns requirements for the `userQuery` stage, and the JWT:

```
curl \
--header "X-OpenIDM-Username: anonymous" \
--header "X-OpenIDM-Password: anonymous" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "input":{}
}' \
"https://localhost:8443/openidm/selfservice/reset?_action=submitRequirements"
{
  "type": "userQuery",
  "tag": "initial",
  "requirements": {
    "$schema": "http:\/\/json-schema.org\/draft-04\/schema#",
    "description": "Find your account",
    "type": "object",
    "required": [
      "queryFilter"
    ],
    "properties": {
      "queryFilter": {
        "description": "filter string to find account",
        "type": "string"
      }
    }
  },
  "token": "eyJ0e…FYkE"
}
```

3. The client provides the requirements for the `userQuery` stage, along with the JWT. The process progresses to the `emailValidation` stage:

```
curl \
--header "X-OpenIDM-Username: anonymous" \
--header "X-OpenIDM-Password: anonymous" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "token": "eyJ0e…FYkE",
  "input": {"queryFilter": "userName eq \"bjensen\""}
}' \
"https://localhost:8443/openidm/selfservice/reset?_action=submitRequirements"
{
  "type": "emailValidation",
  "tag": "validateCode",
  "requirements": {
    "$schema": "http:\/\/json-schema.org\/draft-04\/schema#",
    "description": "Verify emailed code",
    "type": "object",
    "required": [
      "code"
    ],
    "properties": {
      "code": {
        "description": "Enter code emailed",
        "type": "string"
      }
    }
  },
  "token": "eyJ0e…FYkE"
}
```

   The server converts that requirement and token to a URL that is emailed.

4. The user receives an email with the password reset link.

   Clicking the link sends another POST request to the `emailValidation` stage, along with the token, and a `code`:

```
curl \
--header "X-OpenIDM-Username: anonymous" \
--header "X-OpenIDM-Password: anonymous" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/#/passwordreset/&token=eyJ0e…FYkE&code=code"
```

   The process advances to the reset stage and returns its requirements.

5. After email validation, the client submits the new password. The process advances to the reset stage, updates the managed object, and exits:

```
curl \
--header "X-OpenIDM-Username: anonymous" \
--header "X-OpenIDM-Password: anonymous" \
--request POST \
--header "Content-Type: application/json" \
--data {
  "token": "eyJ0e…FYkE",
  "input": {
    "password": "Passw0rd"
  }
} \
"https://localhost:8443/openidm/selfservice/reset?_action=submitRequirements"
{
  "type": "resetStage",
  "tag": "end",
  "status": {
    "success": true
  },
  "additions": {
  }
}
```

Username retrieval

Username retrieval lets registered users retrieve a forgotten username, based on the provision of alternative information in the user record, such as email address, last name, or given name. Depending on how this process is configured, the retrieved username can be emailed to the user or displayed directly.

The REST requests in this section assume that the username is emailed to the user, and that the configuration is similar to that in the example configuration file (`samples/example-configurations/self-service/selfservice-username.json`):

```
{
    "stageConfigs" : [
        {
            "name" : "userQuery",
            "validQueryFields" : [
                "mail",
                "givenName",
                "sn"
            ],
            "identityIdField" : "_id",
            "identityEmailField" : "mail",
            "identityUsernameField" : "userName",
            "identityServiceUrl" : "managed/user"
        },
        {
            "name" : "emailUsername",
            "emailServiceUrl" : "external/email",
            "emailServiceParameters" : {
                "waitForCompletion" : false
            },
            "from" : "info@admin.org",
            "mimeType" : "text/html",
            "subjectTranslations" : {
                "en" : "Account Information - username"
            },
            "messageTranslations" : {
                "en" : "<h3>Username is:</h3><br />%username%"
            },
            "usernameToken" : "%username%"
        },
        {
            "name" : "retrieveUsername"
        }
    ],
    "storage" : "stateless"
}
```

Username retrieval configuration

To set up basic forgotten username configuration, you'll need at least the following configuration files:

- `selfservice-username.json`

  You can find a template version of this file in the following directory: `openidm/samples/example-configurations/self-service`.

- `ui-configuration.json`

  You can find this file in the default IDM project configuration directory, `openidm/conf`.

To set up forgotten username retrieval, enable the following boolean in `ui-configuration.json`:

```
"forgotUsername" : true,
```

You can include several features with forgotten username retrieval, as shown in the following excerpts of the `selfservice-reset.json` file:

- If you've activated Google reCAPTCHA for forgotten username retrieval, you'll refer to the following code block:

```
{
    "name" : "captcha",
    "recaptchaSiteKey" : "<siteKey>",
    "recaptchaSecretKey" : "<secretKey>",
    "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
},
```

  As suggested by the code, you'd substitute actual `siteKey` and `secretKey` assigned by Google for your domain. For more information, refer to Google reCAPTCHA.

- For forgotten username retrieval, IDM needs to verify user identities. To ensure that usernames are sent to the right user, include the following code block:

```
{
    "name" : "userQuery",
    "validQueryFields" : [
        "mail",
        "givenName",
        "sn"
    ],
    "identityIdField" : "_id",
    "identityEmailField" : "mail",
    "identityUsernameField" : "userName",
    "identityServiceUrl" : "managed/user"
},
```

  This code allows IDM to verify user identities by their username, email address, first name (`givenName`), or last name (`sn` , short for surname).

- If you have included email verification, you must configure an outgoing email server. For details about the required addition to `selfservice-registration.json` , refer to Email for forgotten username.

- The following code block, after confirming user identity, allows IDM to display the username:

```
{
    "name" : "retrieveUsername"
}
```

Configuring Forgotten Username Retrieval From the admin UI

To configure forgotten username retrieval using the admin UI, select **Configure > Forgotten Username**. When you activate **Enable Forgotten Username Retrieval**, a **Configure Forgotten Username Form** window displays, and you can specify:

- **Identity Resource**, typically `managed/user` .

- **Advanced Options, Snapshot Token**, typically a JSON Web Token (JWT).

- **Advanced Options, Token Lifetime**, with a default of 300 seconds.

You can also add these settings to the `selfservice-username.json` configuration file. When you modify these settings in the admin UI, IDM creates the file for you.

Email for forgotten username

To configure emails for forgotten username functionality, you can add the following code block to the `selfservice-username.json` file:

```
{
    "name" : "emailUsername",
    "emailServiceUrl" : "external/email",
    "emailServiceParameters" : {
        "waitForCompletion" : false
    },
    "from" : "info@example.com",
    "mimeType" : "text/html",
    "subjectTranslations" : {
        "en" : "Account Information - username"
    },
    "messageTranslations" : {
        "en" : "<h3>Username is:</h3><br />%username%"
    },
    "usernameToken" : "%username%"
},
```

As suggested by the code block, it includes default email messages in English (`en`), with a `usernameToken` that includes the actual username in the message.

As noted in Username retrieval, you can make these changes over the following endpoint URI: `/openidm/config/selfservice/username`

If desired, you can also configure forgotten username retrieval emails through the admin UI. Select **Configure > Forgotten Username**, and click the ✎ button. The **Configure Email Username** window displays.

When you use the admin UI to customize forgotten username retrieval emails, you can review the changes in the `selfservice-username.json` file.

Forgotten username REST requests

The following REST requests and responses demonstrate the flow through a forgotten username process:

1. Client initiates the username retrieval process. The server returns the initial set of requirements:

```
curl \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-Password: anonymous" \
 --header "X-OpenIDM-NoSession: true" \
 --request GET \
 "https://localhost:8443/openidm/selfservice/username"
{
   "_id":"1",
   "_rev":"959264722",
   "type":"userQuery",
   "tag":"initial",
   "requirements":{
      "$schema":"http://json-schema.org/draft-04/schema#",
      "description":"Find your account",
      "type":"object",
      "required":[
         "queryFilter"
      ],
      "properties":{
         "queryFilter":{
            "description":"filter string to find account",
            "type":"string"
         }
      }
   }
}
```

2. Client submits the requirements, along with the token. Server returns the username and the `end` tag to indicate the end of the process:

```
curl \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-Password: anonymous" \
 --request POST \
 --data '{
  "token": "eyJ0eXAiOiJKV1QiLCJjdHkiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ZXlKMGVY…W5ywOcr8",
  {
     "input":{
        "queryFilter":"mail eq \"babs.k.jensen@gmail.com\""
  }
}' \
 "https://localhost:8443/openidm/selfservice/username?_action=submitRequirements"
{
   "type":"retrieveUsername",
   "tag":"end",
   "status":{
      "success":true
   },
   "additions":{
      "userName":"bjensen"
   }
}
```

Additional configuration

This chapter describes additional configuration options for user self-service.

Additional Configuration Options

Email Notification

Configure Notification Emails.

Privacy & Consent

Configure Privacy and Consent.

UMA & Trusted Devices

Set Up User-Managed Access (UMA), Trusted Devices, and Privacy.

Terms & Conditions

Configure Terms & Conditions.

User Self-Service Tokens

Tokens and User Self-Service.

End User UI Notifications

Configure End User UI Notifications.

reCAPTCHA

Configure Google reCAPTCHA.

Identity Fields

Configure Identity Field Associations.

Security Questions

Configure Security Questions (KBA).

Custom Policies

Add Custom Policies for Self-Registration and Password Reset.

End User UI

Configure Self-Service End User UI.

Notification emails

When you configure the outbound email service, IDM can use that service to notify users of significant events, primarily related to user self-service. For specifics, refer to the following table for related notification emails:

Configuring Notification Emails

Situation	Configuration File	Details
When a user is successfully registered	`emailTemplate-welcome.json`	See User Self-Registration Email Template
When a user asks for their forgotten username	`selfservice-username.json`	See Email for forgotten username
When a user registers using self-service and needs to verify their email address	`selfservice-registration.json`	See Self-Service registration emails
When a user asks for a password reset	`selfservice-reset.json`	See Email for password reset

Each email template can specify an email address to use in the `From` field. If this field is left blank, IDM will default to the address specified in Email Settings.

Note

Email templates utilize Handlebar expressions to reference object data dynamically. For example, to reference the `userName` of an object:

```
{{object.userName}}
```

Note

Some email providers, such as Google, will override the `From` address you specify in the templates, and instead use the address used to authenticate with the SMTP server. The email address specified in the template may still be present, but in an email header hidden from most users, such as `X-Google-Original-From`.

User self-registration email template

When a new user registers through the IDM self-registration interface (and if you have configured outbound email), that user will get a welcome email as configured in the `emailTemplate-welcome.json` file:

```
{
    "enabled" : true,
    "from" : "",
    "subject" : {
        "en" : "Your account has been created"
    },
    "message" : {
        "en" : "<html><body><p>Welcome to OpenIDM. Your username is '{{object.userName}}'.</p></body></html>"
    },
    "defaultLocale" : "en"
}
```

You may want to make the following changes:

- Add an email address to the `from` property, perhaps an email address for your organization's systems administrator.
- Specify locale(s) in the `defaultLocale` property.

  Note that locales specified as `preferredLocales` in the `Accept-Language` header take precedence over the `defaultLocale`.
- Modify the subject line as needed.
- Include a welcome `message` appropriate to your organization.

Managing email templates using the admin UI

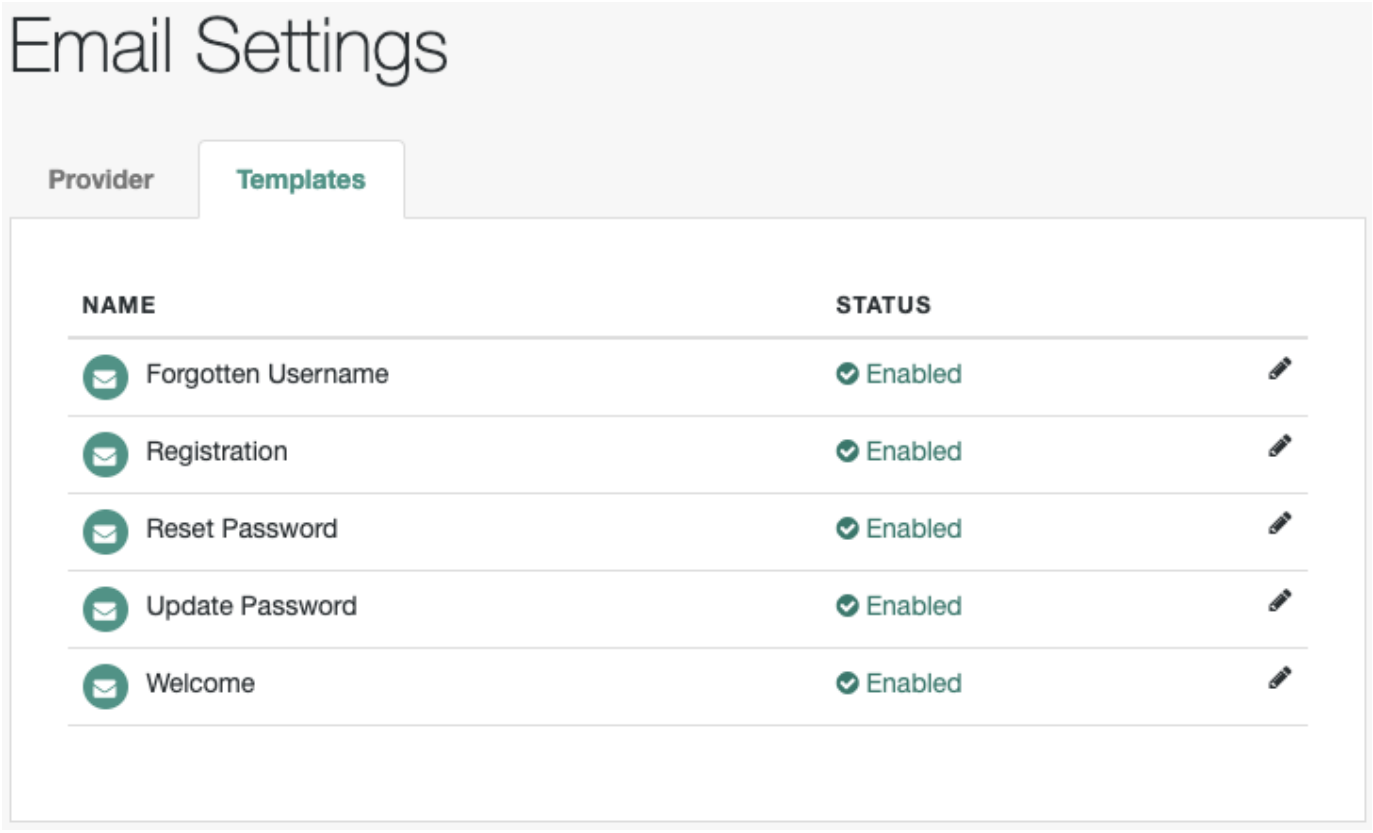
The admin UI includes tools that can help you customize email messages related to the administrative tasks:

- Creating users
- Resetting passwords

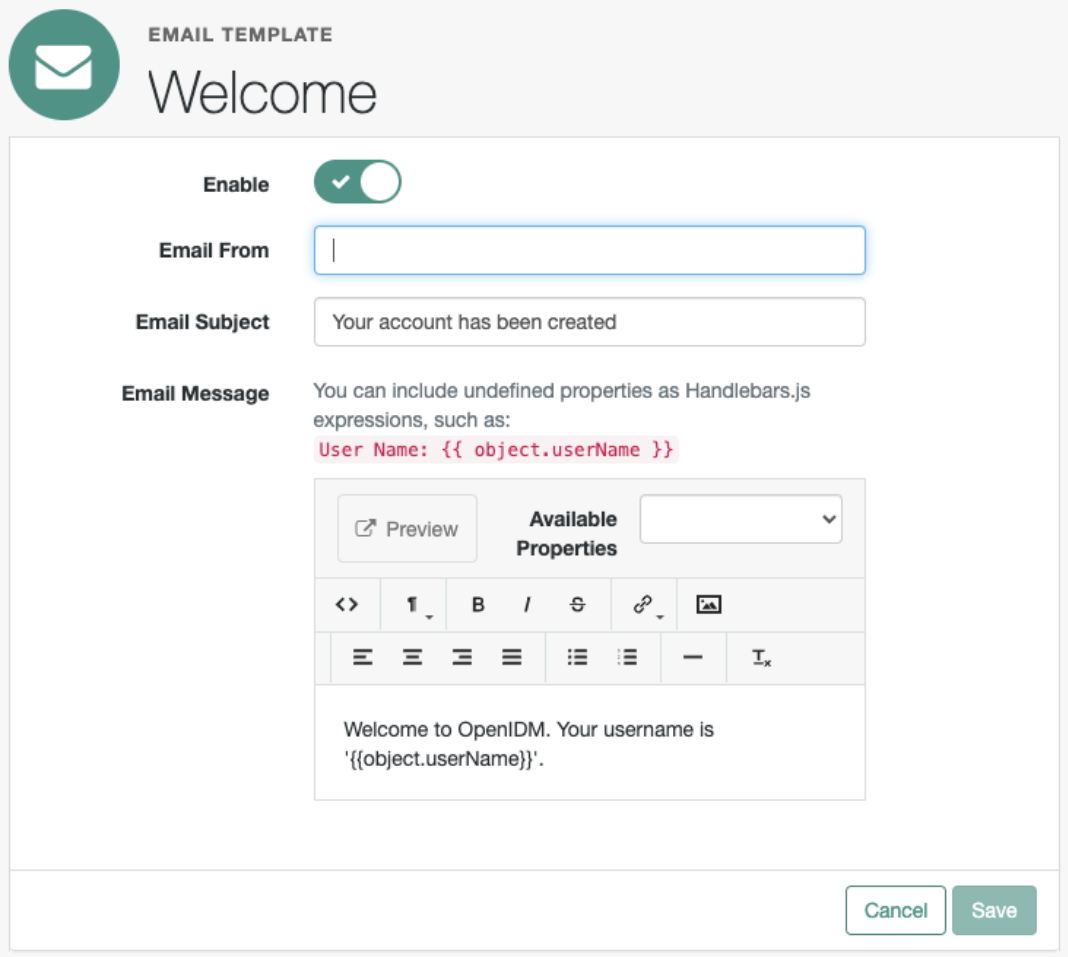
To configure these messages using the admin UI:

1. From the navigation bar, click **Configure > Email Settings**, and select the **Templates** tab.

   IDM displays a list of email templates.



2. To configure an email template, click the adjacent edit button.
3. On the **Email Template templateName** page, make changes, and click **Save**.



Privacy and consent

As an end user, you might want to control what happens to your personal data. For IDM, that means control of how your data is shared with external systems. The example in Marketo connector shows how you can generate a marketing leads database, only for those users who have selected a specific preference. Also read Privacy and consent.

IDM allows you to regulate access to two different kinds of personal data:

- *User information*: while marketers want user information such as addresses and telephone numbers, IDM allows you to let individual users decide whether to share that data. For more information, refer to Regulating HTTP Access to Personal Data.
- *Account information*: by default, IDM prevents REST-based access to passwords with the `private` scope, as defined in the `managed.json` file. You can extend this protection to other properties. For more information, refer to Restricting HTTP Access to Sensitive Data.

You can configure Privacy and Consent for users who register directly through IDM, or through a social identity provider. For more information on the registration process, refer to User self-registration and Social registration.

When you have configured Privacy and Consent, end users must agree to share their data before they can obtain a registered account.

To configure Privacy and Consent, edit the following configuration files:

- In `selfservice-registration.json`, add the following JSON object:

```
{
    "name" : "consent",
    "consentTranslations" : {
     "en" : "<Substitute appropriate Privacy and Consent wording>",
     "fr" : "<Substitute appropriate Privacy and Consent wording, in French>"
    }
},
```

  Add custom privacy and consent notices for all your required languages in the `consentTranslations` property.

  Alternatively, send the corresponding request over REST to the `/openidm/config/selfservice/registration` endpoint.

- In the mapping configuration, include:

```
"consentRequired" : true,
```

Configure privacy and consent using the admin UI

1. From the navigation bar, click **Configure > Mappings**, and select a mapping.

> **Important**
>
> Although the admin UI includes the **Privacy & Consent** switch for all mappings, it makes sense to configure Privacy and Consent *only* for mappings *from* the Managed Object source *to* an external target resource. In other words, end users give their consent to transfer some or all of their managed user data to an external system.

2. Click the **Advanced** tab, activate **Enable Privacy & Consent**, and then click **Save**.
3. From the navigation bar, click **Configure > User Registration**.

> **Note**
>
> If **Enable User Registration** is inactive, refer to User Self-Registration.

4. Click the **Options** tab, and activate **Privacy & Consent**.
5. In the **Configure Privacy & Consent** window, add privacy notices for any necessary languages, and click **Save**.

Regulate HTTP access to personal data

In some cases, you might want to allow users to choose whether to share their personal data. User preferences describes how to allow users to select basic preferences for updates and marketing. They can select these preferences when they register and in the End User UI.

Examine the `managed.json` file for your project. Every relevant property should include two settings that determine whether a user can choose to share or not share that property:

- `isPersonal`: When set to `true`, specifies personally identifying information. By default, the `isPersonal` option for `userName` and `postalAddress` is set to `true`.

  `usageDescription`: Includes additional information that can help users understand the sensitivity of a specific property such as `telephoneNumber`.

The `consentedMappings` property in a managed user object enables the user to specify an array of mappings (target systems) with which they consent to sharing their identifying information. The following sample excerpt of the default managed user object schema shows the `consentedMappings` property definition:

```
"consentedMappings": {
    "title": "Consented Mappings",
    "description": "Consented Mappings",
    "type": "array",
    "viewable": false,
    "searchable": false,
    "userEditable": true,
    "usageDescription": "",
    "isPersonal": false,
    "items": {
      "type": "object",
      "title": "Consented Mapping",
      "properties": {
        "mapping": {
          "title": "Mapping",
          "description": "Mapping",
          "type": "string",
          "viewable": true,
          "searchable": true,
          "userEditable": true
        },
        "consentDate": {
          "title": "Consent Date",
          "description": "Consent Date",
          "type": "string",
          "viewable": true,
          "searchable": true,
          "userEditable": true
        }
      },
      "order": [
        "mapping",
        "consentDate"
      ],
      "required": [
        "mapping",
        "consentDate"
      ]
    },
    "returnByDefault": false,
    "isVirtual": false
}
```

Restrict HTTP access to sensitive data

You can protect specific sensitive managed data by marking the corresponding properties as `private`. Private data, whether it is encrypted or not, is not accessible over the REST interface. Properties that are marked as private are removed from an object when that object is retrieved over REST.

To mark a property as private, set its `scope` to `private` in the `conf/managed.json` file.

The following extract of the `managed.json` file shows how HTTP access is prevented on the `password` property:

```
{
    "objects": [
        {
            "name": "user",
            "schema": {
                "id" : "http://jsonschema.net",
                "title" : "User",
                ...
                "properties" : {
                    ...
                    "password" : {
                        "title" : "Password",
                        ...
                        "encryption" : {
                            "purpose": "idm.password.encryption"
                        },
                        "scope" : "private",
                        ...
        }
    ]
}
```

Tip

To configure private properties using the admin UI:

1. Select **Configure > Managed Objects**, and select the object type whose property values you want to make private (for example, **User**).
2. On the **Properties** tab, select the property that must be private, and select the **Private** checkbox.

A potential caveat relates to private properties. If you use an HTTP `GET` request, you won't even refer to private properties. Even if you know all relevant private properties, a `PUT` request would replace the entire object in the repository. In addition, that require would effectively remove all private properties from the object. To work around this limitation, use a `POST` request to update only those properties that require change.

For example, to update the `givenName` of user jdoe, you could run the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request POST \
--data '[
  {
    "operation": "replace",
    "field": "/givenName",
    "value": "Jon"
  }
]' \
"https://localhost:8443/openidm/managed/user?_action=patch&_queryId=for-userName&uid=jdoe"
```

Note

The filtering of private data applies only to direct HTTP read and query calls on managed objects. No automatic filtering is done for internal callers, and the data that these callers choose to expose.

UMA, trusted devices, and privacy

In the following sections, you will refer to AM documentation to set up User-Managed Access (UMA), Trusted Devices, and Privacy for your end users. The section requires IDM authentication with AM bearer tokens and the `rsFilter` authentication module. For more information, refer to Authenticate through AM.

Tip

If you want to configure both UMA and Trusted Devices in AM, configure these features in the following order, as described in the sections that follow:

1. Set up UMA
2. Use AM to configure UMA-based resources
3. Configure Trusted Devices

If you have to reconfigure UMA at a later date, you'll have to first disable Trusted Devices. You can enable Trusted Devices, once again, afterwards.

User Managed Access in IDM

When you integrate IDM with ForgeRock Access Management (AM) you can take advantage of AM's abilities to work with User-Managed Access (UMA) workflows. AM and IDM use a common installation of ForgeRock Directory Services (DS) to store user data.

When you have configured IDM to authenticate through AM bearer tokens, you can configure AM to work with UMA. For more information, refer to the AM User-Managed Access (UMA) Guide. From that guide, you need to know how to:

- Set up AM as an authorization server.
- Register resource sets and client agents in AM.

- Help users manage access to their protected resources through AM.

Pay close attention to the AM documentation on configuring an OAuth 2.0 UMA Client and UMA Server. You may need to add specific grant types to each OAuth 2.0 application.

If you follow AM documentation to set up UMA, you'll refer to instructions on setting up users as resource owners and requesting parties. If you set up users in AM, be sure to include the following information for each user:

- First Name
- Last Name
- Email Address

AM writes this information to the common DS user data store. You can then synchronize these users to the IDM Managed User data store, with a command such as:

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

After your users have shared UMA resources from the AM Self-Service UI, they can view what they've done and shared in the IDM End User UI, by selecting the Sharing icon (↷).

Configuring Trusted Devices on IDM

You can configure Trusted Devices through AM, using the following sections of the AM Authentication and Single Sign-On Guide: Configuring Authentication Chains and Device ID (Match) Authentication Module. You can use the techniques described in these sections to set up different authentication chains for administrators and regular users.

You can create an AM authentication chain with the following modules and criteria:

AM Authentication Chain Modules

Module	Criteria
Data Store	Requisite
Device Id (Match)	Sufficient
Device Id (Save)	Required

This is different from the authentication chain described in the following section of the *AM Authentication and Single Sign-On Guide*: Device ID (Match) Authentication Module, as it does not include the HOTP Authentication Module.

When trusted devices are enabled, users are presented with a prompt on a screen with the following question "Add to Trusted Devices?". If the user selects `Yes`, that user is prompted for the name of the Trusted Device.

> **Note**
>
> In default configurations, trusted devices are not saved for the AM `amadmin` account. You can set up different AM administrative users as described in Delegate privileges in the AM *Security Guide*.
> You can set up different authentication chains for regular and administrative users, as described in the AM Authentication and Single Sign-On Guide.

Terms & Conditions

Most entities require users to accept Terms & Conditions. By default, this feature is active for user self-registration in IDM. When a user accepts Terms & Conditions, IDM records relevant information in the `_meta` data for that user, as described in Identifying When a User Accepts Terms & Conditions.

> **Note**
>
> To use this feature, `auth.profile.json` must be present in the `/path/to/openidm/conf/` directory.

Terms & Conditions configuration files

`selfservice.terms.json`

Exists in the `/path/to/openidm/conf/` directory and contains the default Terms & Conditions language:

```
{
  "versions": [
    {
      "version": "0.0",
      "termsTranslations": {
        "en": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut
labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore
eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum."
      },
      "createDate": "2019-10-28T04:20:11.320Z"
    }
  ],
  "active": "0.0",
  "uiConfig": {
    "displayName": "We've updated our terms",
    "purpose": "You must accept the updated terms in order to proceed.",
    "buttonText": "Accept"
  }
}
```

`selfservice-termsAndConditions.json`

To force existing IDM users to accept new Terms & Conditions during login, copy `selfservice-termsAndConditions.json` from your project's `conf` directory to your project directory, and edit the file, as necessary.

The following example applies Terms & Conditions to the `managed/user` store:

```
{
  "stageConfigs" : [
    {
      "name" : "conditionaluser",
      "identityServiceUrl" : "managed/user",
      "condition" : {
        "type" : "terms"
      },
      "evaluateConditionOnField" : "user",
      "onConditionTrue" : {
        "name" : "termsAndConditions"
      }
    },
    {
      "name" : "patchObject",
      "identityServiceUrl" : "managed/user"
    }
  ]
}
```

Note

IDM does not support `<form>` elements or `<script>` tags in Terms & Conditions text.
Substitute Terms & Conditions content to meet the legal requirements of your applicable governing entities.

`selfservice.terms.json` *Details*

Property	Description
`version`	Specifies a version number (must be unique).
`termsTranslations`	Supports Terms & Conditions in different languages. **Note** For Terms & Conditions in multiple languages, what the end user sees depends on their browser default language, based on ISO-639 language codes: First, IDM determines the active version, as defined in the `selfservice.terms.json` file: • If the browser default language matches one of the configured Terms & Conditions languages, IDM displays it. • If the browser default language does not match any configured Terms & Conditions languages: ◦ IDM displays the `en` language. ◦ If there is no `en` language, IDM displays the first configured language for the active version.
`createDate`	Creation date.

Property	Description
`active`	Specifies the version of Terms & Conditions shown to users; must match an existing `version`.
`displayName`	The title of the Terms & Conditions page, as seen by end users.
`purpose`	Help text shown below the `displayName`.
`buttonText`	Button text shown to the end user for acceptance.

Preview Terms & Conditions as an end user

To preview Terms & Conditions in the End User UI:

1. Create a regular user.
2. Log in to the End User UI as the new user.

   IDM prompts you to accept the default Terms & Conditions.

Updating Terms & Conditions over REST

You can manage the configuration for Terms & Conditions over the following endpoints:

- `openidm/config/selfservice.terms`
- `openidm/config/selfservice/termsAndConditions`

For example, the following command would replace the value of `buttonText`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
  "operation" : "replace",
  "field" : "uiConfig/buttonText",
  "value" : "OK"
} ]' \
"http://localhost:8080/openidm/config/selfservice.terms"
```

Identifying when a user accepts Terms & Conditions

You can identify when a user accepts Terms & Conditions, as well as the associated version. To do so, take the following steps:

- If needed, find identifying information for all managed users:

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

- Use REST to get a specific user's information. This example illustrates how a user with a `userName` of `kvaughan` has already accepted a specific version of Terms & Conditions:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=userName+eq+'kvaughan'&_fields=*,/_meta/*"
{
  "result": [
    {
      ...
      "userName": "kvaughan",
      ...
        "termsAccepted": {
          "acceptDate": "2018-04-12T22:55:33.370Z",
          "termsVersion": "2.0"
        },
        "createDate": "2018-04-12T22:55:33.395Z",
        "lastChanged": {
          "date": "2018-04-12T22:55:33.395Z"
        },
        "loginCount": 1,
        "_rev": "00000000776f8be1",
        "_id": "69124007-05ec-46e1-a8a8-ecc3d94db124"
      }
    }
  ],
  ...
}
```

Configure Terms & Conditions using the admin UI

> **Note**
>
> The admin UI does not let you delete existing Terms & Conditions.

1. From the navigation bar, click **Configure > Terms & Conditions**.
2. Click **New Version**, and on the **New Terms & Conditions Version** page, configure the following:
   - **Version** (must be unique).

- If there are existing Terms & Conditions, a **Make active** switch displays. If you activate this option, all users must accept the new, active Terms & Conditions.
- **Locale**, in ISO-639 format.
- **Terms & Conditions**, in the specified language locales. You can set up Terms & Conditions in text and/or basic HTML.

3. After you've added Terms & Conditions to all desired locales, click **Save**.

   IDM saves your changes in the `selfservice.terms.json` file.

4. Once you have at least one set of Terms & Conditions, click the **Settings** tab, configure the following, and click **Save**:
   - **Require acceptance** switch—the next time any end user logs into IDM, that user will refer to a copy of your Terms & Conditions, with the **Header**, **Description**, and **Button Text**.
   - **Header**.
   - **Description**.
   - **Button Text**.
5. To make sure new users must accept the Terms & Conditions:
   1. From the navigation bar, click **Configure > User Registration**, and select the **Options** tab.
   2. Enable Terms & Conditions. For more information, refer to Self-registration.

These changes are recorded in `_meta` data for each user, and can be retrieved through REST calls described in Identifying When a User Accepts Terms & Conditions.

Tokens and user self-service

Many processes within user self-service involve multiple stages, such as user self-registration, password reset, and forgotten username. As the user transitions from one stage to another, IDM uses JWT tokens to represent the current state of the process. As each stage is completed, IDM returns a new token. Each request that follows includes that latest token.

For example, users who use these features to recover their usernames and passwords get two tokens in the following scenario:

- The user goes through the forgotten username process, gets a JWT Token with a lifetime (default = 300 seconds) that lets the user get to the next step in the process.
- With username in hand, that user may then start the password reset process. That user gets a second JWT token, with the token lifetime configured for that process.

> **Note**
>
> The default IDM JWT token is encrypted and stateless. However, if you need a token that can be included in a link that works in all email clients, change the `snapshotToken `type` in the appropriate configuration file to `uuid`.

End User UI notifications

Whenever there are changes related to individual users, IDM sends notifications to those users. When they log in to the End User UI, they can find their notifications by clicking the notification 🔔 button.

Notifications are configured in `notification-event.json` files, as described in Custom Notifications.

IDM includes a `notifications` endpoint that can help you identify all notifications:

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

To list notifications by user ID, include the `_notifications` field in a query on that ID:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/e3a9385b-733f-4a1c-891b-c89292b30d70?_fields=_notifications/*"
```

You can filter notifications with any of the properties shown in the following table:

End User Notification Properties

Property	Description
`createDate`	Creation date
`notificationType`	Message type: limited to info, warning, or error
`message`	Message seen by the end user

You can get additional information from the activity audit log, in the `audit/activity.audit.json` file, including the following:

- The `userId` who made the change.
- The `runAs` name of the user who made the change.
- If configured in Fields to Watch, any watched fields that have changed.
- If the password was changed, as indicated by the `passwordChanged` property.

Google reCAPTCHA

Google reCAPTCHA helps prevent bots from registering users or resetting passwords on your system. For Google documentation on this feature, refer to Google reCAPTCHA. IDM works with Google reCAPTCHA v2.

To use Google reCAPTCHA, you will need a Google account and your domain name (RFC 2606-compliant URLs such as `localhost` and `example.com` are acceptable for test purposes). Google then provides a site key and a secret key that you can include in the self-service function configuration.

For example, you can set up reCAPTCHA by adding the following code block to the configuration file for user self-registration `selfservice-registration.json`, password reset, `selfservice-reset.json`, and forgotten username `selfservice-username.json` functionality.

```
{
    "name" : "captcha",
    "recaptchaSiteKey" : "< Insert Site Key Here >",
    "recaptchaSecretKey" : "< Insert Secret Key Here >",
    "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
},
```

You may also add the reCAPTCHA keys through the UI for each of these self-service features.

Identity fields

It is possible to adjust the property associated with a field in user self-service. Properties that are used by self-service functions can be set using identity field properties in your configuration. For example, if you had changed the `mail` property in `managed/user` to instead be `email`, you would then update `identityEmailField` in your self-service configuration to be `"identityEmailField" : "email",`. There are currently six identity fields that can be customized:

- `identityServiceUrl` - sets where self-service stores and retrieves its data, such as `managed/user`.
- `identityUsernameField` - sets the property associated with the username of the user.
- `identityEmailField` - sets the property associated with the email address of the user.
- `identityPasswordField` - sets the property associated with the password of the user.
- `identityIdField` - sets the property associated with the ID of the user, which is used when performing user queries.
- `identityAccountStatus` - sets the property associated with the account status of the user, which is used when performing user queries.

Not every identity field is used in each self-service stage. For more information about which fields are required for each stage, refer to Self-service stage reference.

> **Note**
>
> If you have removed usernames from your `managed/user` schema in favor of using another property (such as email), you will still need to set `identityUsernameField` to the new property in order for self-service to function correctly.

Security questions

IDM uses security questions to let users verify their identities. Security questions are sometimes referred to as Knowledge-Based Authentication (KBA). When an administrator has configured security questions, self-service users can choose from the questions set in the `selfservice.kba.json` file, as described in Security Questions and Self-Registration.

You can prompt users to update their security questions. As these questions may be subject to risks, you can set up IDM to prompt the user to update and/or add security questions, courtesy of the `selfservice-kbaUpdate.json` file. For more information, refer to Prompt to Update Security Questions.

Security questions and self-registration

The user is prompted to enter answers to pre-configured or custom security questions, during the self-registration process. These questions are used to help verify an identity when a user requests a password reset. These questions do not apply for users who need username retrieval.

The template version of the `selfservice.kba.json` file includes `minimumAnswersToDefine`, which requires a user to define at least that many security questions and answers, along with `minimumAnswersToVerify`, which requires a user to answer (in this case), at least one of those questions when asking for a password reset.

```
{
    "kbaPropertyName" : "kbaInfo",
    "minimumAnswersToDefine": 2,
    "minimumAnswersToVerify": 1,
    "questions" : {
        "1" : {
            "en" : "What's your favorite color?",
            "en_GB" : "What is your favourite colour?",
            "fr" : "Quelle est votre couleur préférée?"
        },
        "2" : {
            "en" : "Who was your first employer?"
        }
    }
}
```

You can change or add questions in JSON format, or if you're configuring user self-registration, you can also edit these questions through the admin UI. From the admin UI, select **Configure > User Registration**. Enable User Registration, select **Options > Security Questions**, and select the edit icon to add, edit, or delete these questions.

Any change you make to the security questions under User Registration also applies to Password Reset. To confirm, select **Configure > Password Reset**. Enable Password Reset, and edit the Security Questions. You'll refer to the same questions there.

In addition, individual users can configure their own questions and answers:

- During the user self-registration process.
- From the End User UI, in the user's Profile section (), under **Account Security > Security Questions**.

Important

A managed user's security questions can only be changed through the `selfservice/userupdate` endpoint, or when the user is created through `selfservice/registration`, and provides their own questions. You cannot manipulate a user's `kbaInfo` property directly through the `managed/user` endpoint.
When the answers to security questions are hashed, they are converted to lowercase. If you intend to pre-populate answers with a mapping, the `openidm.hash` function, or the `secureHash` mechanism, you must provide the string in lowercase to match the value of the answer.

KBA answer hashing

By default, KBA answers are SHA-256 hashed upon save. To specify another type of hashing, edit the self-service KBA configuration]]>:

Add the secureHash property to the `conf/selfservice.kba.json` file:

```
"secureHash" : {
  "algorithm": "{type}",
  "configProp": value
}
```

For example, to use BCRYPT hashing:

```
"secureHash": {
  "algorithm": "BCRYPT",
  "cost": 13
}
```

1. Get the current self-service KBA configuration]]>:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/config/selfservice.kba"
{
  "_id": "selfservice.kba",
  "kbaPropertyName": "kbaInfo",
  "minimumAnswersToDefine": 2,
  "minimumAnswersToVerify": 1,
  "questions": {
    "1": {
      "en": "What’s your favorite color?",
      "en_GB": "What is your favourite colour?",
      "fr": "Quelle est votre couleur préférée?"
    },
    "2": {
      "en": "Who was your first employer?"
    }
  }
}
```

2. Add the `secureHash` property for the alternative hashing, and replace the self-service KBA configuration]]>. For example, to use BCRYPT hashing:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "_id": "selfservice.kba",
  "kbaPropertyName": "kbaInfo",
  "minimumAnswersToDefine": 2,
  "minimumAnswersToVerify": 1,
  "questions": {
    "1": {
      "en": "What'\''s your favorite color?",
      "en_GB": "What is your favourite colour?",
      "fr": "Quelle est votre couleur préférée?"
    },
    "2": {
      "en": "Who was your first employer?"
    }
  },
  "secureHash": {
    "algorithm": "BCRYPT",
    "cost": 13
  }
}' \
"http://localhost:8080/openidm/config/selfservice.kba"
{
  "_id": "selfservice.kba",
  "kbaPropertyName": "kbaInfo",
  "minimumAnswersToDefine": 2,
  "minimumAnswersToVerify": 1,
  "questions": {
    "1": {
      "en": "What’s your favorite color?",
      "en_GB": "What is your favourite colour?",
      "fr": "Quelle est votre couleur préférée?"
    },
    "2": {
      "en": "Who was your first employer?"
    }
  },
  "secureHash": {
    "algorithm": "BCRYPT",
    "cost": 13
  }
}
```

Supported Hashing Algorithms and Configuration Properties

Algorithm	Config Property and Description
`BCRYPT`	**`cost`** Value between 4 and 31. Default is `13`.
`PBKDF2`	**`hashLength`** Byte-length of the generated hash. Default is `16`. **`saltLength`** Byte-length of the salt value. Default is `16`. **`iterations`** Number of cryptographic iterations. Default is `20000`. **`hmac`** HMAC algorithm. Default is `SHA3-256`. Supported values: • `SHA-224` • `SHA-256` • `SHA-384` • `SHA-512` • `SHA3-224` • `SHA3-256` • `SHA3-384` • `SHA3-512`
`SCRYPT`	**`hashLength`** Byte-length of the generated hash, must be greater than or equal to 8. Default is `16`. **`saltLength`** Byte-length of the salt value. Default is `16`. **`n`** CPU/Memory cost. Must be greater than 1, a power of 2, and less than *2^(128 * r / 8)*. Default is `32768`. **`p`** Parallelization. Must be a positive integer less than or equal to *Integer.MAX_VALUE / (128 * r * 8)*. Default is `1`. **`r`** Block size. Must be greater than or equal to 1. Default is `8`.

Algorithm	Config Property and Description
`SHA-256`	**`saltLength`** Byte-length of the salt value. Default is `16`. **Note** This is the default hashing.
`SHA-384`	**`saltLength`** Byte-length of the salt value. Default is `16`.
`SHA-512`	**`saltLength`** Byte-length of the salt value. Default is `16`.

KBA attempts account lockout

To configure account lockout based on the security questions, add the following lines to your `selfservice.kba.json` file:

```
"numberOfAttemptsAllowed" : 2,
"kbaAttemptsPropertyName" : "lockoutproperty"
```

With this configuration, users who make more than two mistakes in answering security questions are prevented from using the password reset facility until the `kbaAttemptsPropertyName` field is removed, or the number is set to a value lower than the `numberOfAttemptsAllowed`. The number of mistakes is recorded in whatever property you assign to `kbaAttemptsPropertyName` (`lockoutproperty`, in this example).

If you are using an explicit mapping for managed user objects, you must add this lockoutproperty to your database schema *and* to the `objectToColumn` mapping in your repository configuration file.

For example, the previous configuration would require the following addition to your `conf/repo.jdbc.json` file:

```
"explicitMapping" : {
    "managed/user": {
        "table" : "managed_user",
        "objectToColumn": {
            ...
            "lockoutproperty" : "lockoutproperty",
            ...
        }
```

You would also need to create a `lockoutproperty` column in the `openidm.managed_user` table, with datatype `VARCHAR`. For example:

```
mysql> show columns from managed_user;
+---------------------+--------------+------+-----+---------+-------+
| Field               | Type         | Null | Key | Default | Extra |
+---------------------+--------------+------+-----+---------+-------+
| objectid            | varchar(38)  | NO   | PRI | NULL    |       |
| rev                 | varchar(38)  | NO   |     | NULL    |       |
| username            | varchar(255) | YES  | UNI | NULL    |       |
| password            | varchar(511) | YES  |     | NULL    |       |
| accountstatus       | varchar(255) | YES  | MUL | NULL    |       |
| postalcode          | varchar(255) | YES  |     | NULL    |       |
| lockoutproperty     | varchar(255) | YES  |     | NULL    |       |
...
```

> **Warning**
>
> Once you deploy these IDM self-service features, you should never remove or change existing security questions, as users may have included those questions during the user self-registration process.

Prompt to update security questions

IDM supports a requirement for users to update their security questions, in the `selfservice-kbaUpdate.json` file. You can find this file in the following directory: `/path/to/openidm/samples/example-configurations/self-service`.

Alternatively, if you set up security questions from the admin UI, you can navigate to Configure > Security Questions > Update Form, and select Enable Update. This action adds a `selfservice-kbaUpdate.json` file to your project's `conf/` subdirectory.

For more information on this configuration file, refer to Conditional User Stage.

Custom policies for self-registration and password reset

IDM defines policies for usernames and passwords, in the `openidm/bin/defaults/script/policy.js` file. To enforce these policies for user self-registration and password reset, add the following objects to your `conf/policy.json` file, under `resources`:

```
{
    "resource" : "selfservice/registration",
    "calculatedProperties" : {
        "type" : "text/javascript",
        "source" : "require('selfServicePolicies').getRegistrationProperties()"
    }
},
{
    "resource" : "selfservice/reset",
    "calculatedProperties" : {
        "type" : "text/javascript",
        "source" : "require('selfServicePolicies').getResetProperties()"
    }
},
```

Self-service end user UI

This topic includes procedures to verify functionality from an end user point of view. Some options described can be used to help support compliance with the General Data Protection Regulation (GDPR).

For information about customizing the End User UI, refer to the Github repository: ForgeRock/end-user-ui.

Localize the end user UI

The End User UI is configured in US English. For more information on how to localize and modify the messages in the End User UI, refer to Translations and Text.

Change the end user UI path

By default, the End User UI is registered at the root context and is accessible at the URL `https://localhost:8443`. To specify a different URL, edit the `project-dir/conf/ui.context-enduser.json` file, setting the `urlContextRoot` property to the new URL.

For example, to change the End User UI URL to `https://localhost:8443/exampleui`, edit the file as follows:

```
"urlContextRoot" : "/exampleui",
```

Alternatively, to change the End User UI URL in the admin UI, follow these steps:

1. Log in to the admin UI.
2. From the navigation bar, click **Configure > System Preferences**, and select the **Self-Service UI** tab.
3. Specify the new context route in the **Relative URL** field.
4. Click **Save**.

Provide a logout URL to external applications

By default, an End User UI session is invalidated when a user clicks on the Log out link. In certain situations, external applications might require a distinct logout URL to which users can be routed, to terminate their UI session.

The logout URL is `#logout`, appended to the UI URL; for example, `https://localhost:8443/#logout/`.

The logout URL effectively performs the same action as clicking on the Log out link of the UI.

Privacy: my account information in the End User UI

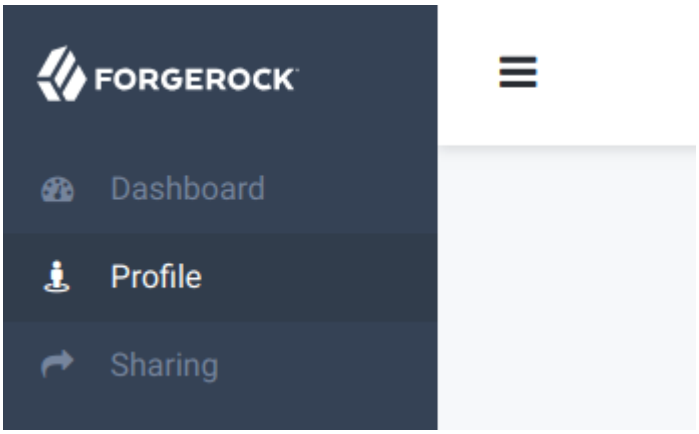
While end users can find their information in the End User UI, you can use REST calls and audit logs to find the same information. Some of the information in this section, such as Trusted Devices and UMA-based sharing, might require integration with ForgeRock Access Management (AM), as described in the Platform Setup Guide.

What the end user sees upon log in to the End User UI depends on which features are configured.

- When you log in to the End User UI, you'll be taken to the IDM Profile page, with at least the following information under the **Settings** tab:
  - **Account Security**
  - **Preferences**

- **Account Controls**

- At a minimum, the left panel displays the **Dashboard** and **Profile** buttons. If you've configured UMA as described in UMA, trusted devices, and privacy, you'll also refer to a Sharing button. To see descriptions, click the Menu button:



- When you add features, additional options display on the profile page:

Information in the End User Profile Page

Title	Description	Section
Account Security	Password and Security Questions, default	Security questions
Social Sign-in	Links to Social Identity Provider Accounts	Social registration
Authorized Applications	Applications that can access an account	Authorized Applications
Trusted Devices	Based on system and browser	Configuring Trusted Devices on IDM
Preferences	Default	User preferences
Personal Data Sharing	Provides control	Personal Data Sharing
Account Controls	Includes collected account data (Default)	Account Controls

Personal information

To view account details in the End User UI, a user clicks the **Profile button > Edit Personal Info**. By default, user information includes at least a Username, First Name, Last Name, and Email Address.

Each user can modify this information as needed, as long as `"userEditable" : true` for the property in your project's `managed.json` file. For more information, refer to Create and Modify Object Types.

Sign-In & security

Under this tab, end users can change their passwords. They can also add, delete, or modify security questions, and link or unlink supported social identity accounts. For more information, refer to Security questions and Social registration.

Preferences

The preferences tab allows end users to modify marketing preferences, as defined in the `managed.json` file, and the Managed Object User property Preferences tab. For more information, refer to Configure User Preferences.

End users can toggle marketing preferences. When IDM includes a mapping to a marketing database, these preferences are sent to that database. This can help administrators use IDM to target marketing campaigns and identify potential leads.

Trusted devices

A *trusted device* uses AM's Device ID (Match) and Device ID (Save) authentication modules, as described in the AM Authentication and Single Sign-On Guide. When such modules are configured (see Configuring Trusted Devices on IDM), end users can add such devices the first time they log in from a new location.

During the login process, when an end user selects *Log In*, that user is prompted for a *Trusted Device Name*. Users refer to their added devices under the Trusted Devices tab.

A trusted device entry is paired with a specific browser on a specific system. The next time the same end user logs in from the same browser and system, in the same location, that user should not be prompted to enter a trusted device again.

End users can remove their trusted devices from the tab.

Authorized applications

The Authorized Applications section is specific to end users as OAuth 2 clients. and reflects the corresponding section of the AM Self-Service dashboard, as described in the following section of the AM *OAuth 2.0 Guide on*: User Consent Management.

Personal data sharing

This section assumes that as an administrator, you've followed the instructions in Privacy and consent to enable Privacy & Consent.

End users who refer to a Personal Data Sharing section have control of whether personal data is shared with an external database, such as one that might contain marketing leads.

The managed object record for end users who consent to sharing such data is shown in REST output and the audit activity log as one `consentedMappings` object:

```
"consentedMappings" : [ {
   "mapping" : "managedUser_systemLdapAccounts",
   "consentDate" : "2017-08-25T18:13:08.358Z"
}
```

If enabled, end users will refer to a Personal Data Sharing section in their profiles. If they select the Allow link, they can see the data properties that would be shared with the external database.

This option supports the right to restrict processing of user personal data.

Account controls

The Account Controls section allows end users to download their account data (in JSON format), and to delete their accounts from IDM.

> **Important**
>
> When end users delete their accounts, the change is propagated to external systems by implicit sync. However, it is then up to the administrator of the external system to make sure that any additional user information is purged from that system.

To modify the message associated with the `Delete Your Account` option, refer to the section about Translations in the README of the public ForgeRock Identity Management (End User) Git repository. Find the `translation.json` file, search for the `deleteAccount` code block, and edit the information.

The options shown in this section can help meet requirements related to data portability, as well as the right to be forgotten.

Custom self-service stages

This chapter demonstrates how to build, deploy, and configure a custom stage, and how to add it to a self-service process. You can use the classes in the sample project as a basis to develop your own stages.

To implement a custom stage in the End User UI, refer to the following instructions from the ForgeRock End User UI Git Repository: How to Add a Self-Service Stage to the UI.

Sample stage

ForgeRock provides a sample custom stage project with the minimum classes and project file required for any self-service stage. The sample project has a dependency on the `forgerock-selfservice-core` artifact. Engage ForgeRock support for access to the required repositories.

The sample project implements a stage named `MathProblem`, which generates a simple math problem that must be completed in order to progress to the next stage.

The project includes the following files, required for any custom self-service stage:

A Maven project file (`pom.xml`)

Pay particular attention to the `maven-bundle-plugin` in this file:

```
<plugins>
    <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
            <instructions>
                <Fragment-Host>org.forgerock.openidm.selfservice</Fragment-Host>
            </instructions>
        </configuration>
    </plugin>
</plugins>
```

This plugin indicates that Apache Felix should attach the custom stage artifact to IDM's self-service bundle.

A configuration class

(`src/main/java/org/forgerock/selfservice/custom/MathProblemStageConfig.java`)

The configuration class reads configuration data from a corresponding configuration (JSON) file. The class represents each configuration item for the stage as properties of the class.

An implementation class

(`src/main/java/org/forgerock/selfservice/custom/MathProblemStage.java`)

The implementation class is the main orchestration class for the stage.

Build the sample stage

To build the sample stage, you must have Apache Maven installed.

1. Clone the ForgeRock Selfservice Custom Stage repository.
2. Change to the root directory of the project you cloned:

```
cd /path/to/forgerock-selfservice-custom-stage
```

3. This version of IDM works with version `26.3.x` of ForgeRock Commons. Locate the latest version `26.3.x` tag:
    1. List the latest tags for this version:

```
git tag --list | grep 26.3.x
26.3.x-20210331172848-d863e5b
...
26.3.x-latest
```

    2. Check out the latest version:

```
git checkout -b test tags/26.3.x-latest
Switched to a new branch 'test'
```

4. Build the sample stage:

```
mvn clean install
```

The build process creates the `forgerock-selfservice-custom-stage/self-service/forgerock-selfservice-custom-stage/target/forgerock-selfservice-custom-stage-version.jar` file.

5. Copy the compiled sample stage to the `openidm/bundle` directory:

```
cp target/forgerock-selfservice-custom-stage-version.jar /path/to/openidm/bundle/
```

6. Restart IDM.

Configuration for the sample stage

To create a configuration for this stage, examine the configuration class (`MathProblemStageConfig.java`). Three configuration properties must be specified in the corresponding configuration file:

- `class`

  For the default IDM self-service stages, you specify the stage `name` in the configuration, in the format `"name" : "stage-name"` . For example:

  ```
  "name" : "captcha"
  ```

  For custom stages, you must specify the stage configuration class, in the format `"class" : "stage_config_classname"` . For example:

  ```
  "class" : "org.forgerock.selfservice.custom.MathProblemStageConfig"
  ```

- `leftValue`
- `rightValue`

The configuration for this stage will therefore look something like the following:

```
{
    "class" : "org.forgerock.selfservice.custom.MathProblemStageConfig",
    "leftValue" : int,
    "rightValue" : int
},
```

> **Important**
>
> When you write a custom stage, the `equals` and `hashCode` methods must be overridden to include local class members.

Test the custom stage

Stages are implemented as part of a self-service process. To test your custom stage, you need to add it to a self-service process. You can create a new process, or use one of the default processes available through the admin UI.

In this example, we add the custom stage to the User Registration process and test it as part of self-registration, as follows:

1. From the navigation bar, click **Configure > User Registration**, and activate **Enable User Registration.**

   IDM creates a `selfservice-registration.json` file in your project's `conf` directory. There are a number of stages in that process by default; for example, the `parameters` stage:

   ```
   "stageConfigs" : [
       {
           "name" : "parameters",
           "parameterNames" : [
               "returnParams"
           ]
       },
   ...
   ]
   ```

2. Add your custom stage to the process by creating a configuration item in the `stageConfigs` array:

   ```
   "stageConfigs" : [
       {
           "name" : "parameters",
           "parameterNames" : [
               "returnParams"
           ]
       },
       {
       "class" : "org.forgerock.selfservice.custom.MathProblemStageConfig",
       "leftValue" : 12,
       "rightValue" : 4
       },
   ...
   ]
   ```

   > **Note**
   >
   > Self-service stages can generally not be configured in random order. For example, some stages require input from the process `state` that has been populated by a preceding stage. For the purposes of this example, add the `MathProblem` stage directly after the `parameters` stage.

3. Disable *all-in-one* registration.

By default, the registration phase has *all-in-one* registration enabled. All-in-one registration covers a number of registration stages. For the purposes of testing the custom stage, disable all-in-one registration by setting `"allInOneRegistration" : false` in `selfservice-registration.json`. For more information, refer to All-in-one registration.

4. Save the changes to the `selfservice-registration.json` file.

   IDM reloads the configuration automatically—you do not need to restart the server.

5. Log in to the End User UI (`https://localhost:8443` by default), and click **Register**.

   IDM displays the Math Problem you configured previously.

Self-service stage reference

This chapter describes the individual stages that can be called by a self-service process, the purpose of the stage, any required parameters, dependencies on preceding or following stages, and the expected stage output.

The stages are listed in alphabetical order, for ease of reference, but they cannot be configured in random order. For example, some stages require input from the process `state` that has been populated by a preceding stage.

The `identityServiceURL` is a required parameter for most self-service stages. The self-service stages operate on a managed object. The `identityServiceURL` indicates the object type, for example, `managed/user`.

All-in-one registration

A registration process that consists of more than one stage can include an optional *"super stage"* named `allInOneRegistration`, that is set outside of the `stageConfigs` array as follows:

```
"allInOneRegistration" : true
```

All-in-one registration covers a number of registration stages. If this property is `true`, in the registration process configuration, IDM scans the configuration for any of the following stages:

- `parameters`
- `captcha`
- `termsAndConditions`
- `kbaSecurityAnswerDefinitionStage`
- `consent`
- `idmUserDetails`

If any of these stages are found, the individual stages are effectively removed from the configuration, and a new configuration is generated that accumulates all the found stages.

The purpose of all-in-one registration is to obtain a set of initial requirements, then to advance to the end of all six stages simultaneously. This lets self-registration be completed on a single registration form. As the process advances, it gathers any output, errors, and others from all six stages (or however many stages have been configured). The process then returns whatever was gathered from the cumulative stages, including any outstanding requirements. Depending on the output, the process might be required to go through the stages more than once, as the outstanding requirements are provided.

Important

All-in-one registration requires multiple registration stages. If your registration process includes only one stage, for example, `consent`, `allInOneRegistration` must be set to `false`, to preserve the registration flow.
If all-in-one registration is `false`, any additional stages listed in the registration process (`selfservice-registration.json`) must be listed *after* the parameters and `idmUserDetails` stages. If a stage occurs before the `idmUserDetails` stage without all-in-one registration, both social and regular registration will not work.

OpenAM auto-login stage

This stage is used to perform auto-login when IDM is configured with ForgeRock Access Management (AM). The stage is similar to the local auto-login stage, but also requires the `returnParams` stored in `state` (populated in the Parameters stage).

Example configuration

```
{
    "name" : "openAmAutoLogin",
    "identityUsernameField": "userName",
    "identityPasswordField": "password",
    "openAMBaseUrl" : "http://AM.example.com:8080/openam/",
    "authenticationEndpoint" : "json/realms/root/authenticate"
}
```

Dependencies

This stage should appear towards the end of a process—it cannot be the first stage in a process.

Required Parameters

- `authenticationEndpoint` - the AM Authentication Endpoint URL.
- `openAMBaseUrl` - the URL of the AM server.
- `identityUsernameField` - the managed object property that contains the username.
- `identityPasswordField` - the managed object property that contains the user password.

Attribute collection stage

The purpose of this stage is to collect managed object properties to insert into the user profile. The list of properties to be collected is defined as part of the configuration.

This stage updates the managed object directly, and checks whether attributes are *required*. If required attributes are not provided, the stage returns the list of requirements again. This stage can throw an exception if there is an error attempting to save the updated attributes.

Example configuration

```
{
    "name" : "attributecollection",
    "identityServiceUrl" : "managed/user",
    "uiConfig" : {
        "displayName" : "Add your telephone number",
        "purpose" : "Help us verify your identity",
        "buttonText" : "Save"
    },
    "attributes" : [
        {
            "name" : "telephoneNumber",
            "isRequired" : true
        }
    ]
}
```

Dependencies

No dependencies on previous or following stages. This stage can occur anywhere in a process.

Required Parameters

- `identityServiceUrl` - the managed object type on which this stage acts
- `uiConfig` - how the requirements list is conveyed to an end user
- `attributes` - the array of attributes to be collected. For each attribute, the `isRequired` parameter indicates whether the attribute is mandatory for the stage to proceed.

Captcha stage

This stage verifies a `response` variable populated in `state` by the reCaptcha mechanism. If the response is missing, or if validation fails (typically, if the configuration does not include the required reCaptcha configuration parameters), the stage throws a bad request exception. If validation succeeds, the process advances to the next stage.

Example configuration

```
{
    "name" : "captcha",
    "recaptchaSiteKey" : "6LdahVIUAAAAAJcwGTWdl4OsG9tpdgFIyZKUSzyU",
    "recaptchaSecretKey" : "6LdahVIUAAAAANF-O17E-b8PyBqLrhLaOHUX8ch-",
    "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
},
```

Dependencies

No dependencies on previous or following stages. This stage can occur anywhere in a process.

Required Parameters

- `recaptchaSiteKey` - invokes the reCAPTCHA service

- `recaptchaSecretKey` - authorizes communication between IDM and the reCAPTCHA server to verify the user's response
- `recaptchaUri` - the reCaptcha verification API

Conditional User Stage

Defines a condition, that results in a boolean (`true` or `false`). The outcome of the condition determines which stage should be executed next.

Example configuration

```
{
    "name": "conditionaluser",
    "identityServiceUrl": "managed/user",
    "condition": {
        "type": "kbaQuestions"
    },
    "evaluateConditionOnField": "user",
    "onConditionFalse": {
        "name": "kbaUpdateStage",
        "kbaConfig": null,
        "identityServiceUrl" : "managed/user",
        "uiConfig" : {
            "displayName" : "Update your security questions",
            "purpose" : "Please review and update your security questions",
            "buttonText" : "Update"
        }
    }
}
```

Dependencies

No dependencies on previous or following stages. This stage can occur anywhere in a process. If the condition evaluates to `true`, the process moves on to the next stage.

Required Parameters

- `identityServiceUrl` - the managed object type on which this stage acts.
- `condition` - the condition type, which can be one of the following:
  - `kbaQuestions` - a boolean (`true` or `false`) that indicates whether configured security questions have been answered.
  - `queryFilter` - a common filter expression such as `"filter" : "/co eq \"US\""`.
  - `script` - lets you configure a custom scripted condition.
  - `loginCount` - a condition based on the number of password or social authentication-based login requests.
  - `terms` - a boolean (`true` or `false`) that indicates whether configured Terms and Conditions have been accepted.

- `timesincelogin` - sets a condition based on the period of time since the last login, in years, months, weeks, days, hours, and minutes.

- `evaluateConditionOnField` - the property on which the condition should be evaluated.
- `onConditionFalse` - the details of the stage to be called if the condition evaluates to false.

Consent Stage

This stage evaluates a boolean `consentGiven` (`true` or `false`). The user is prompted to consent for each mapping that is set to require consent. If consent is required but not given, the stage fails with an exception. It is up to the client to handle that exception, for example, to prevent registration if the user does not provide consent.

Dependencies

No dependencies on previous or following stages. This stage can occur anywhere in a process.

Required Parameters

- None.

Email validation stage

This stage retrieves the email address from `state` (or in response to initial requirements), then verifies the validity of the email address with the user who submitted the requirements through an email process.

Example configuration

```
{
    "name" : "emailValidation",
    "identityEmailField" : "mail",
    "emailServiceUrl" : "external/email",
    "emailServiceParameters" : {
        "waitForCompletion" : false
    },
    "from" : "info@admin.org",
    "subject" : "Reset password email",
    "mimeType" : "text/html",
    "subjectTranslations" : {
        "en" : "Reset your password",
        "fr" : "Réinitialisez votre mot de passe"
    },
    "messageTranslations" : {
        "en" : "Click to reset your password <a href=\"%link%\">Password reset link</a>",
        "fr" : "Cliquez pour réinitialiser votre mot de passe<a href=\"%link%\">Mot de passe lien de
réinitialisation</a>"
    },
    "verificationLinkToken" : "%link%",
    "verificationLink" : "https://localhost:8443/#/passwordreset/"
},
```

Dependencies

This stage expects a preceding stage to populate the user email address in `state`. The stage has no downstream dependencies.

Required Parameters

- Email configuration. For more information, refer to Self-Service registration emails.

IDM user details stage

This stage collects new user data and stores it in `state`. This is the only stage that sets up a user from nothing. The stage does not *create* a managed object directly—it simply gathers and stores the data. The Self-registration stage consumes the stored user data and creates the managed object from it.

The IDM User Details stage executes multiple times, requesting additional requirements each time. There are different ways for the stage to advance, depending on how the user create request is initiated.

If the user completes a self-service registration form, the input contains a `user` object, collected from the form, and populates that user in `state`. If the user registers through social authentication, the stage reads the profile from the remote identity provider, normalizes it, then maps it to a user object. That user object is then put into `state`.

If the new user object in `state` is incomplete or does not meet policy requirements, the stage returns a new set of requirements, indicating the collected data and the missing data. The registering user is requested to submit the additional data, then the stage revalidates the object in `state`. When all of the required data to register a user is present, the process advances to the next stage.

> **Important**
>
> The user data remains in `state` —no managed user object is created.

Example configuration

```
{
    "name" : "idmUserDetails",
    "identityEmailField" : "mail",
    "socialRegistrationEnabled" : true,
    "identityServiceUrl" : "managed/user",
    "registrationProperties" : [
        "userName",
        "givenName",
        "sn",
        "mail"
    ],
    "registrationPreferences": ["marketing", "updates"]
},
```

Dependencies

This stage *must* occur in any registration process. It has no dependencies on previous stages but must have the Self-registration stage somewhere downstream in the process, to create the managed user object.

Required Parameters

- `identityEmailField` - the attribute on the managed user object that contains the user email.
- `identityServiceUrl` - the managed object type on which this stage acts.
- `socialRegistrationEnabled` - optional, `false` if not specified. Indicates whether the stage must read the user profile from a remote identity provider and normalize it.
- `registrationProperties` - an array of properties that must be provided by a registering user in order for the stage to progress.
- `registrationPreferences` - optional, an array of properties that can be requested after the user has provided the required properties.

KBA security answer definition stage

In the context of registration, this stage supplies security questions to the user and captures the answers provided by the user.

The stage validates any answers against the user object. If the requirement is not met (incorrect number of questions answered correctly), the stage throws a bad request exception and increments the failure count of the managed user. If the requirement is met (correct number of questions answered correctly), the process advances to the next stage.

This stage also disallows users from entering custom questions that duplicate any questions defined by the administrator, regardless of the locale. It does this comparison by removing any special characters and making a lowercase comparison. For example, `What Is YoUr FaVorite COLOR????` would be evaluated as the same question as `what is your favorite color?`.

Example configuration

```
{
    "name" : "kbaSecurityAnswerDefinitionStage",
    "kbaConfig" : null
},
```

Dependencies

The stage depends on a previous stage to populate the user ID in `state`. It has no dependencies on following stages.

Required Parameters

- `kbaConfig` - reads the KBA configuration from the corresponding `selfservice.kba.json` file.

KBA security answer verification stage

This stage verifies security answers and validates user lockout. The stage requires a user ID in `state`.

The stage reads the user object and validates that the user has not already failed to answer the security questions. The stage then obtains the configured security questions, and returns the minimum number of randomly selected questions as a requirement.

The stage validates any answers against the user object. If the requirement is not met (incorrect number of questions answered correctly) the stage throws a bad request exception and increments the failure count of the managed user. If the requirement is met (correct number of questions answered correctly) the process advances to the next stage.

Example configuration

```
{
    "name" : "kbaSecurityAnswerDefinitionStage",
    "kbaConfig" : null
},
```

Dependencies

The stage depends on a previous stage to populate the user ID in `state`. It has no dependencies on following stages.

Required Parameters

- `kbaConfig` - reads the KBA configuration from the corresponding `selfservice.kba.json` file.

KBA update stage

The KBA Update stage is used as part of progressive profile completion to let users update their existing security questions and to add any additional questions that are needed. This stage updates the user object directly. If a user fails to provide sufficient questions, the stage returns the requirements again. If the object cannot be updated, the stage throws an exception. The stage outputs nothing to the `state` and has no downstream dependencies.

Example configuration

```
{
    "name": "kbaUpdateStage",
    "kbaConfig": null,
    "identityServiceUrl" : "managed/user",
    "uiConfig" : {
        "displayName" : "Update your security questions",
        "purpose" : "Please review and update your security questions",
        "buttonText" : "Update"
    }
}
```

Dependencies

No dependencies on previous or following stages. This stage can occur anywhere in a process. If the condition evaluates to `true`, the process moves on to the next stage.

Required Parameters

- `kbaConfig` - returns the minimum number of security questions that must be provided.
- `identityServiceUrl` - the managed object type on which this stage acts.
- `uiConfig` - how the requirements are conveyed to an end user.

Local auto-login stage

This stage is used to perform auto-login with IDM. The stage obtains the `OAuth Login` from `state`, and populates the `user` object (`username` and `password`) in `state`.

The stage adds the OAuth login to the `successAdditions` (with a value of `true`) and adds the `successURL` from its own configuration. If IDM can obtain all those details from `state`, it takes the user object, locates the `username` and `password`, and generates a `CREDENTIAL_JWT`. That JWT is then placed in the `successAdditions` parameter.

If IDM is unable to generate the `CREDENTIAL_JWT`, it generates an internal server error (500).

Example configuration

```
{
    "name" : "localAutoLogin",
    "successUrl" : "",
    "identityUsernameField": "userName",
    "identityPasswordField": "password"
}
```

Dependencies

This stage should appear towards the end of a process—it cannot be the first stage in a process.

Required Parameters

- `successURL` - the URL to which an end user should be redirected following successful registration.
- `identityUsernameField` - the managed object property that contains the username.
- `identityPasswordField` - the managed object property that contains the user password.

Parameters stage

This stage captures parameters in the original request. To advance, the stage assesses the input body. Any values that have been passed in and are listed in the configuration are put into `state`. The stage ignores any values that are not listed in the configuration. The self-service mechanism passes the parameters back to the client at the end of the process.

By default, this stage is required *only* if you are integrating IDM with AM. The stage is added automatically if you use the UI to configure a self-service process, but can generally be ignored unless a custom client or UI requires it.

Example configuration

```
{
    "name" : "parameters",
    "parameterNames" : [
        "returnParams"
    ]
}
```

Dependencies

In all of the default IDM self-service processes, this must be the first stage in the process. In a custom process, the stage has no order dependencies, and can occur anywhere in a process. All this stage does is to copy named parameters into `successAdditions` for the process to output at `tag:end`.

Required Parameters

- `parameterNames` - a list of parameters the stage supports. These parameters are returned in the requirements.

Patch object stage

Currently, this stage is used *only* to patch the managed object with the terms and conditions acceptance obtained from `state`. If the terms and conditions state is not present, the stage simply advances to the next stage in the process.

Example configuration

```
{
    "name" : "patchObject",
    "identityServiceUrl" : "managed/user"
}
```

Dependencies

This stage requires the Terms and Conditions stage to have preceded it. It can be followed by any stage and can occur anywhere in a process.

Requirements

- `identityServiceUrl` - the managed object type on which this stage acts.

Password reset stage

This stage updates the managed object directly, changing the value of the configured `identityPasswordField`. To gather the initial requirements, the stage reads the managed user object, and checks that the `email` and `userID` of the object match what is in `state`. If they do not match, the stage exits with a `Bad request exception`.

If they do match, the stage returns with its requirements (the new `password` value). When the requirements are submitted, the stage advances, locates the `userId` again, and applies the new `password`. If the password is empty, the stage throws an exception. If the password is valid, the stage patches the managed user object directly to update the password. If the patch fails, the stage returns the requirements again, along with an error message (for example, a password policy requirement).

Example configuration

```
{
    "name" : "resetStage",
    "identityServiceUrl" : "managed/user",
    "identityPasswordField" : "password"
}
```

Dependencies

This stage cannot be the first stage in a process. It expects a previous stage to populate the `userId` and `mail` attributes of the `user` in `state`.

Required Parameters

- `identityServiceUrl` - the managed object type on which this stage acts.
- `identityPasswordField` - the managed object property that contains the user password.

Self-registration stage

This is currently the final stage in the default user registration process. The stage obtains all the user details from `state`. When the stage advances, it checks `state` for any `idpdata`, combines that with the user data, and creates the managed user object. This stage *must* occur in any registration process.

> **Note**
>
> If you are integrating IDM with AM, the OpenAM auto-login stage can follow this stage.

Example configuration

```
{
    "name" : "selfRegistration",
    "identityServiceUrl" : "managed/user"
},
```

Dependencies

This stage *must* come after a stage that has populated the user in `state`. If the user is absent, the stage exits with an illegal argument exception.

Required Parameters

- `identityServiceUrl` - the managed object type that the stage creates.

Social user claim stage

> **Important**
>
> Social authentication is deprecated and will be removed in a future release of IDM. For more information, refer to Deprecation.

This stage enables an existing managed user to claim a social identity. The stage obtains a `CLIENT_TOKEN` from some social identity provider. That token includes the following data:

- OAuth token
- Identity provider name
- Renewal token
- Expiration date

Using the `CLIENT_TOKEN`, the stage retrieves the user profile from the social identity provider and normalizes the profile into a user object (using the regular normalization mapping for social identity providers). For more information on this mapping, refer to Many social identity providers, one schema.

If the stage is unable to retrieve the user profile, or unable to normalize it using the mapping, it exits with an exception. It does not return any missing requirements.

When the user profile has been normalized, the stage attempts to identify any existing managed users that match the profile. If there are no matches, it simply advances to the next stage in the process. If it finds a match, it extracts the existing managed object and returns that as a new set of requirements.

The new requirement is that the user must provide their `password`, either their managed/user password, or the password to another social identity provider, if they registered through a separate identity provider.

The stage then does the following:

- Verifies the login
- Creates a `managed/idp` object for the user
- Establishes a relationship between the managed object and the idp object
- Puts `OAUTH_LOGIN:true` into `state`
- Puts a `claimedProfile` containing the URL of the managed object that was claimed into `successAdditions`

Example configuration

```
{
    "name" : "socialUserClaim",
    "identityServiceUrl" : "managed/user",
    "claimQueryFilter" : "/mail eq \"{{mail}}\""
},
```

Dependencies

This stage has no dependencies on previous or subsequent stages and can occur anywhere in a process.

Required Parameters

- `identityServiceUrl` - the managed object type against which the stage verifies the profile.
- `claimQueryFilter` - the query filter that is used to locate the managed object from the social identity provider profile.

  Notice the double-brace notation in preceding example `"claimQueryFilter" : "/mail eq \"{{mail}}\""`. This notation indicates that the named property from the user object in `state` is substituted for the double-braced value. In this example, `{{mail}}` would become the value of the `mail` property of the user in `state`, such as `bjensen@example.com`, if that was in the user in `state`. You can use this notation with any user property.

Terms and Conditions stage

This stage evaluates a boolean `accepted` (`true` or `false`).

Example configuration

This stage is configured in a `selfservice.terms.json` file in the project `conf` directory and includes the following parameters:

```
{
    "versions" : [
        {
            "version" : "1",
            "termsTranslations" : {
                "en" : "Sample terms and conditions"
            },
            "createDate" : "2018-04-10T09:52:25.478Z"
        }
    ],
    "uiConfig" : {
        "displayName" : "We have updated our terms",
        "purpose" : "To proceed, accept these terms",
        "buttonText" : "Accept"
    },
    "active" : "1"
}
```

The stage can stand on its own (as it does in the default registration configuration) or be called from the Conditional User Stage with a configuration similar to the following:

```
{
    "name" : "conditionaluser",
    "identityServiceUrl" : "managed/user",
    "condition" : {
        "type" : "terms"
    },
    "evaluateConditionOnField" : "user",
    "onConditionTrue" : {
        "name" : "termsAndConditions"
    }
},
```

Dependencies

Configured as part of the Conditional User Stage. *Must* have the Patch object stage somewhere downstream. This stage can occur anywhere in a process.

Requirements

Requires Terms and Conditions to be accepted before continuing to the next stage:

- If `accept` is absent, the stage returns the requirements again.
- If `accept` is present but `false`, the stage generates an exception. It is up to the client to handle that exception.
- If `accept` is `true`, this stage puts all the outputs into `state` and advances to the next stage.

Outputs

`TERMS_ACCEPTED`, `TERMS_DATE`, and `TERMS_VERSION`

User query stage

This stage queries the managed user repository for a user, based on the supplied query fields. If the stage identifies a user, it populates the `mail`, `userId`, `userName`, and `accountStatus` fields in `state`.

Example configuration

```
{
    "name" : "userQuery",
    "validQueryFields" : [
        "userName",
        "mail",
        "givenName",
        "sn"
    ],
    "identityIdField" : "_id",
    "identityEmailField" : "mail",
    "identityUsernameField" : "userName",
    "identityServiceUrl" : "managed/user",
    "identityAccountStatusField" : "accountStatus"
},
```

Dependencies

This stage has no dependencies on preceding or following stages, but cannot be the only stage in a process.

Required Parameters

- `validQueryFields` - an array of fields on which the query can be based.
- `identityIdField` - the managed object property that contains the user ID to be provided to `state`.
- `identityEmailField` - the managed object property that contains the user mail to be provided to `state`.
- `identityUsernameField` - the managed object property that contains the username to be provided to `state`.
- `identityAccountStatusField` - the managed object property that contains the user account status to be provided to `state`.
- `identityServiceUrl` - the managed object type on which this stage acts.

IDM glossary

correlation query

A correlation query specifies an expression that matches existing entries in a source repository to one or more entries in a target repository. A correlation query might be built with a script, but it is not the same as a correlation script. For more information, refer to Correlate source objects with existing target objects.

correlation script

A correlation script matches existing entries in a source repository, and returns the IDs of one or more matching entries on a target repository. While it skips the intermediate step associated with a `correlation query`, a correlation script can be relatively complex, based on the operations of the script.

entitlement

An entitlement is a collection of attributes that can be added to a user entry via roles. As such, it is a specialized type of `assignment`. A user or device with an entitlement gets access rights to specified resources. An entitlement is a property of a managed object.

JCE

Java Cryptographic Extension, which is part of the Java Cryptography Architecture, provides a framework for encryption, key generation, and digital signatures.

JSON

JavaScript Object Notation, a lightweight data interchange format based on a subset of JavaScript syntax. For more information, refer to the JSON site.

JSON Pointer

A JSON Pointer defines a string syntax for identifying a specific value within a JSON document. For information about JSON Pointer syntax, refer to the JSON Pointer RFC.

JWT

JSON Web Token. As noted in RFC 8725, "JSON Web Tokens, also known as JWTs, are URL-safe JSON-based security tokens that contain a set of claims that can be signed and/or encrypted." For IDM, the JWT is associated with the `JWT_SESSION` authentication module.

managed object

An object that represents the identity-related data managed by IDM. Managed objects are configurable, JSON-based data structures that IDM stores in its pluggable repository. The default configuration of a managed object is that of a user, but you can define any kind of managed object, for example, groups or roles.

mapping

A policy that is defined between a source object and a target object during reconciliation or synchronization. A mapping can also define a trigger for validation, customization, filtering, and transformation of source and target objects.

OSGi

A module system and service platform for the Java programming language that implements a complete and dynamic component model. For more information, refer to What is OSGi? Currently, only the Apache Felix container is supported.

reconciliation

During reconciliation, comparisons are made between managed objects and objects on source or target systems. Reconciliation can result in one or more specified actions, including, but not limited to, synchronization.

resource

An external system, database, directory server, or other source of identity data to be managed and audited by the identity management system.

REST

Representational State Transfer. A software architecture style for exposing resources, using the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed.

role

IDM distinguishes between two distinct role types - provisioning roles and authorization roles. For more information, refer to Managed Roles.

source object

In the context of reconciliation, a source object is a data object on the source system, that IDM scans before attempting to find a corresponding object on the target system. Depending on the defined mapping, IDM then adjusts the object on the target system (target object).

synchronization

The synchronization process creates, updates, or deletes objects on a target system, based on the defined mappings from the source system. Synchronization can be scheduled or on demand.

system object

A pluggable representation of an object on an external system. For example, a user entry that is stored in an external LDAP directory is represented as a system object in IDM for the period during which IDM requires access to that entry. System objects follow the same RESTful resource-based design principles as managed objects.

target object

In the context of reconciliation, a target object is a data object on the target system, that IDM scans after locating its corresponding object on the source system. Depending on the defined mapping, IDM then adjusts the target object to match the corresponding source object.