Audit Guide

/ ForgeRock Identity Management 7

Latest update: 7.0.3

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

Abstract

Guide to configuring audit logs and reports.

Table of Contents

Overview

This guide covers audit logging and reporting.

Quick Start

 Audit Configure audit logging.	 Reporting Configure audit reports and 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, see https://www.forgerock.com.

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

Chapter 1

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, you'll see an audit log file for each configured audit event topic 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 on 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.splunk.SplunkAuditEventHandler",
            "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, Syslog, and Splunk 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 basic configuration properties. Specific audit event handlers have additional configuration properties.

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.

+ *List the Active Audit Event Handlers*

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, see "*Audit Event Handler Configuration*".

- "JSON Audit Event Handler"
- "JSON Standard Output Audit Event Handler"
- "CSV Audit Event Handler"

- "Router Audit Event Handler"
- "Repository Audit Event Handler"
- "JMS Audit Event Handler"
- "Syslog Audit Event Handler"
- "Splunk Audit Event Handler"

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",
            "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, see "Specify the Startup Configuration" in the *Installation Guide*.

You can use property value substitution to direct log files to another location on the filesystem. For more information, see "Property Value Substitution" in the *Setup Guide*.

`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, see "Viewing Reconciliation Association Details" in the *Synchronization Guide*.

For a description of all the configurable properties of the JSON audit event handler, see "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" : {
        "elasticsearchCompatible" : true,
        "name" : "StdOut",
        "topics" : [
            "config",
            "activity",
            "authentication",
            "access",
            "recon",
            "sync"
        ],
        "enabled" : true
    }
}...
```

Audit file names are fixed and correspond to the event being audited:

`access.audit.json`
`activity.audit.json`
`authentication.audit.json`
`config.audit.json`
`recon.audit.json`

`sync.audit.json`

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, see "Specify the Startup Configuration" in the *Installation Guide*.

You can use property value substitution to direct logs to another location on the filesystem. For more information, see "Property Value Substitution" in the *Setup Guide*.

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 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:

+ *Configure Keys to Protect Audit Logs*

  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. Select 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:

+ *Check Log File Integrity*

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, you'll see a message similar to:

```
    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" in the *Samples Guide*. 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.jdbc.Driver",
        "url" : "jdbc:mysql://&{openidm.repo.host}:&{openidm.repo.port}/audit",
        "autoCommit" : true,
        "jdbcDriver" : "com.mysql.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*. For a demonstration, see "*Direct Audit Information to a JMS Broker*" in the *Samples Guide*.
- *TIBCO Enterprise Message Service*, as described in this chapter.

This implementation supports the *publish/subscribe* model. For more information, see *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, see 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 ActiveMQ, and a number of dependencies. This section lists the dependencies, where you can download them, and where they must be installed in the IDM instance. If you use a different ActiveMQ version, the dependency versions may differ from those shown.

Download the following files:

- ActiveMQ binary. This sample was tested with ActiveMQ 5.15.13.
- ActiveMQ Client that corresponds to your ActiveMQ version.
- JmDNS, version 3.4.1.
- The most recent `bnd` JAR file from https://repo1.maven.org/maven2/biz/aQute/bnd/biz.aQute.bnd/. The bnd utility lets you create OSGi bundles for libraries that do not yet support OSGi.
- Apache Geronimo J2EE management bundle.
- hawtbuf-1.11 JAR file (a Maven-based protocol buffer compiler).

1. Unpack the ActiveMQ binary. For example:

```
tar -zxvf ~/Downloads/apache-activemq-5.15.13-bin.tar.gz
```

2. Create a temporary directory, move the ActiveMQ Client, and `bnd` JAR files to that directory, then change to that directory:

```
mkdir ~/Downloads/tmp
mv activemq-client-5.15.13.jar ~/Downloads/tmp/
mv biz.aQute.bnd-version.jar ~/Downloads/tmp/
cd ~/Downloads/tmp/
```

3. Create an OSGi bundle as follows:

   a. In a text editor, create a BND file named `activemq.bnd` and save it to the current directory. The file should have the following contents:

```
version=5.15.13
Export-Package: *;version=${version}
Bundle-Name: ActiveMQ :: Client
Bundle-SymbolicName: org.apache.activemq
Bundle-Version: ${version}
```

   Your `tmp/` directory should now contain the following files:

```
ls ~/Downloads/tmp/
activemq-client-5.15.13.jar activemq.bnd   biz.aQute.bnd-version.jar
```

b. In that same directory, create the OSGi bundle archive file as follows:

```
java -jar biz.aQute.bnd-version.jar \
wrap --properties activemq.bnd \
--output activemq-client-5.15.13-osgi.jar \
activemq-client-5.15.13.jar
```

4. Copy the resulting `activemq-client-5.15.13-osgi.jar` file to the `openidm/bundle` directory:

```
cp activemq-client-5.15.13-osgi.jar /path/to/openidm/bundle/
```

5. Copy the *Apache Geronimo*, *hawtbuf*, and *JmDNS* JAR files to the `openidm/bundle` directory:

```
cp ~/Downloads/geronimo-j2ee-management_1.1_spec-1.0.1.jar /path/to/openidm/bundle/
cp ~/Downloads/hawtbuf-1.11.jar /path/to/openidm/bundle/
cp ~/Downloads/jmdns-3.4.1.jar /path/to/openidm/bundle
```

Your IDM instance is now ready for you to configure the JMS audit event handler.

Configure 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, see "JMS Audit Event Handler Properties".

To configure the audit event handler in the `conf/audit.json` file, see 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.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 Java Naming and Directory Interface (JNDI) message broker, on port 61616. For an example of how to configure Apache ActiveMQ, see "*Direct Audit Information to a JMS Broker*" in the *Samples Guide*.

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 ActiveMQ

This configuration provides a connection to the ActiveMQ server instance with TLSv1.3.

1. In the directory where you unpacked ActiveMQ, edit the `conf/activemq.xml` file as follows:

    - In the `<brokers>` element, add an `<sslContext>`:

```
<broker xmlns="http://activemq.apache.org/schema/
core" brokerName="localhost" dataDirectory="${activemq.data}">
    ...
    <sslContext>
        <sslContext keyStore="file:${activemq.conf}/broker.ks" keyStorePassword="password"/>
    </sslContext>
    ...
</broker>
```

- In the `<transportConnectors>` element, add an `ssl` `<transportConnector>`:

```
<transportConnectors>
    ...
    <transportConnector name="ssl" uri="ssl://0.0.0.0:61617?transport.needClientAuth=false"/>
</transportConnectors>
```

Note

To enable mutual authentication, set `transport.needClientAuth=true`, and import the IDM server certificate into the ActiveMQ truststore (`conf/broker.ts`).

2. Delete the existing self-signed server certificate from the ActiveMQ keystore and truststore:

```
keytool \
-delete \
-keystore /path/to/activeMQ/conf/broker.ts \
-alias broker-localhost
Enter keystore password: password
keytool \
-delete \
-keystore /path/to/activeMQ/conf/broker.ks \
-alias broker-localhost
Enter keystore password: password
```

3. Generate a new self-signed server certificate for ActiveMQ:

```
keytool \
-genkey \
-keyalg RSA \
-alias broker-localhost \
-keystore /path/to/activeMQ/conf/broker.ks \
-storepass password \
-validity 360 \
-keysize 2048
```

Important

The `CN` in the generated self-signed certificate *must* match the hostname that you specify in the IDM JMS provider URL. If you are using `localhost` to connect to the broker, you must specify `localhost` when **keytool**

prompts you for the `first and last name`. If the `CN` is not the same as the hostname, the server certificate validation will fail.

4. Export the ActiveMQ server certificate:

```
keytool \
-export \
-alias broker-localhost \
-file broker-localhost.key \
-keystore /path/to/activeMQ/conf/broker.ks
Enter keystore password: password
Certificate stored in file <broker-localhost.key>
```

5. Import the ActiveMQ server certificate into the IDM truststore:

```
keytool \
-import \
-alias activemq \
-keystore /path/to/openidm/security/truststore \
-file broker-localhost.key
Enter keystore password: changeit
Owner: CN=localhost, OU=Unknown, O=example.com, L=Unknown, ST=Unknown, C=Unknown
Issuer: CN=localhost, OU=Unknown, O=example.com, L=Unknown, ST=Unknown, C=Unknown
...
Trust this certificate? [no]: yes
Certificate was added to keystore
```

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 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, see 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, see "Syslog Audit Event Handler Properties".

Splunk Audit Event Handler

The Splunk audit event handler logs IDM events to a Splunk system.

Important

Currently, the Splunk audit event handler can only be used to write events to Splunk. It cannot read or query audit events. You must therefore use the Splunk audit event handler in tandem with another event handler that is configured to handle queries.

Splunk enables you to define the structure of the incoming data. To use the event handler with IDM, create a new data Source Type in Splunk that will be associated with the incoming IDM log data. Because the audit event handler users the HTTP endpoints in Splunk, you must also enable a Splunk HTTP Event Collector. The HTTP Event Collector provides an authorization token that allows IDM to log events to Splunk.

The following procedure assumes a Splunk instance running on the same host as IDM. Adjust the instructions for your Splunk system:

1. Create a new source type:

   a. In the Splunk UI, select Data > Source Types > New Source Type.

   b. Provide a name for the source type, for example, `openidm`.

   c. Under Event Breaks, specify how the incoming messages are split.

      The Splunk audit event handler supports bulk handing, so it passes multiple audit events to Splunk at a time, as a large JSON payload.

Select Regex and enter `^{` to indicate how the bulk messages are separated.

d. Under Timestamp, click Auto to specify that Splunk should generate the timestamp, then click Save.

2. Create a new HTTP Event Collector.

a. Select Data Inputs > HTTP Event Collector > New Token.

b. Enter a Name that will be associated with this token, for example, `openidm`.

Other fields are optional.

c. On the Input Settings screen, click Select under Source Type, then select Custom > openidm from the Select Source Type list.

d. Click Review, then Submit.

Important

Splunk provides the authorization token that you must add as the value of the `authzToken` property in the Splunk audit event handler configuration.

e. Make sure that the Global Settings for HTTP Event Collectors do not conflict with the settings you have configured for this IDM HTTP Event Collector.

To add the Splunk audit event handler to your IDM configuration, update your project's `audit.json` file or select Configure > System Preferences > Audit in the Admin UI, then select SplunkAuditEventHandler and click Add Event Handler.

The following excerpt of an `audit.json` file shows a sample Splunk audit event handler configuration. Adjust the connection settings and `authzToken` to match your Splunk system.

```
{
    "class" : "org.forgerock.audit.handlers.splunk.SplunkAuditEventHandler",
    "config" : {
        "connection" : {
            "useSSL" : false,
            "host" : "localhost",
            "port" : 8088
        },
        "buffering" : {
            "maxSize" : 10000,
            "writeInterval" : "100 ms",
            "maxBatchedEvents" : 500
        },
        "authzToken" : "87E9C00F-F5E6-47CF-B62F-E415A8142355",
        "name" : "Splunk",
        "topics" : [
            "config",
            "activity",
            "authentication",
            "access",
            "recon",
            "sync"
        ],
        "enabled" : true
    }
}
```

All properties are mandatory. For a complete list of the configurable properties for this audit event handler, see "Splunk 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, see "*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, add the following line to 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.)
	`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:

Event Type	Actions	Description
		`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, see "*Synchronization Situations and Actions*" in the *Synchronization Guide*.
	`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 enables you to 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, see "Synchronization Actions" in the *Synchronization Guide*.

Use Policies to Filter Audit Data

In addition to event-based filtering, you can use policies to select the specific information to include 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*

+ *Access Safelist*

- /_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

+ *Activity Safelist*

- /_id
- /timestamp
- /eventName
- /transactionId
- /trackingIds
- /userId
- /runAs
- /objectId
- /operation
- /changedFields
- /revision
- /status
- /message

- /passwordChanged
- /context
- /provider

+ *Authentication Safelist*

- /_id
- /timestamp
- /eventName
- /transactionId
- /trackingIds
- /userId
- /principal
- /entries
- /result
- /provider
- /method

+ *Configuration Safelist*

- /_id
- /timestamp
- /eventName
- /transactionId
- /trackingIds
- /userId
- /runAs
- /objectId
- /operation
- /changedFields

- /revision

+ *Reconciliation Safelist*

- /_id
- /action
- /ambiguousTargetObjectIds
- /entryType
- /eventName
- /exception
- /linkQualifier
- /mapping
- /message
- /messageDetail
- /reconAction
- /reconciling
- /reconId
- /situation
- /sourceObjectId
- /status
- /targetObjectId
- /timestamp
- /trackingIds
- /transactionId
- /userId

+ *Synchronization Safelist*

- /_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.

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 appears 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, see "*Schedule Synchronization*" in the *Synchronization Guide*.

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 enable you to 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, you will see two `access.audit.json` files 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.

You can also set up an aggregated analysis of audit logs over REST on the `report/audit` endpoint. For more information, see "Generate Audit Reports".

Note

Queries on the audit endpoint must use queryFilter syntax.

If you get no REST output on the correct endpoint, there might be no audit data in the corresponding audit file or JDBC table.

Some of the 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.

+ *Query the Reconciliation Audit Log*

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"
    }
  ],
  ...
}
```


+ *Query the Activity Audit Log*

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.

+ *Query the Synchronization Audit Log*

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"
 }
```

+ *Query the Authentication Audit Log*

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 see 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, see "Authentication and Session Modules" in the *Security Guide*.

Login failures can also be instructive, as you'll see 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`.

+ *Query the Configuration Audit Log*

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

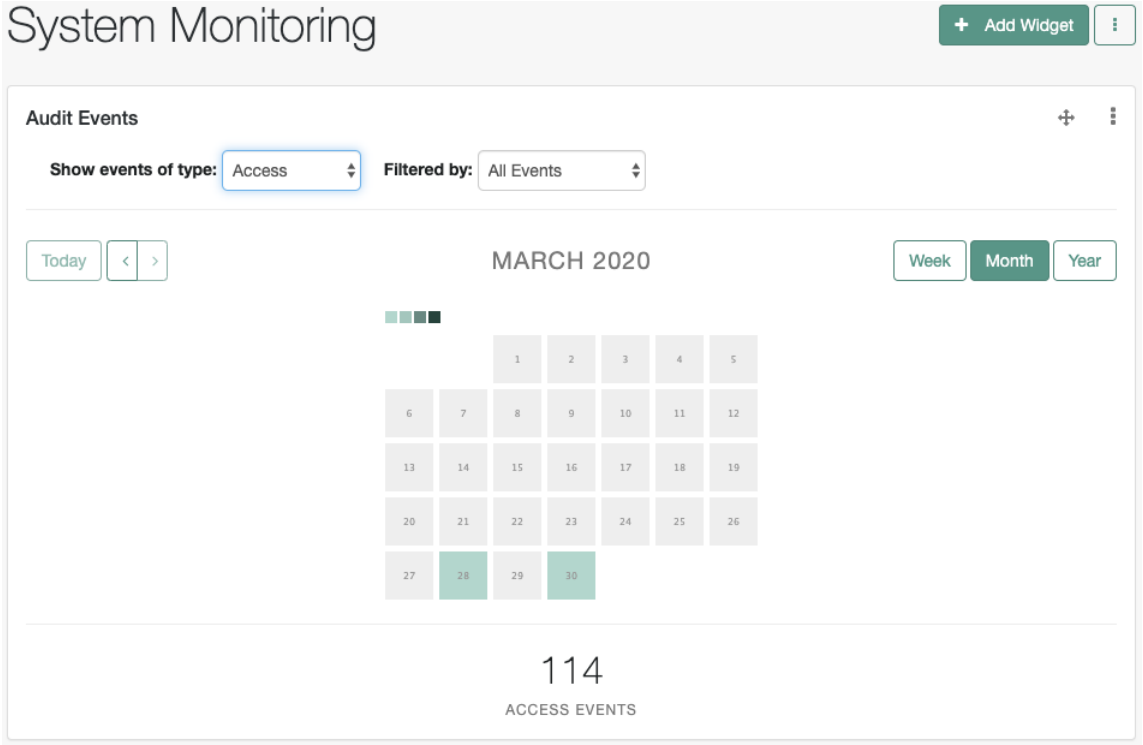
The Admin UI includes an audit widget that provides a visual display of audit events.

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.

Chapter 2

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"`.
`transactionId`	UUID of the transaction; you might see the same ID 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"`.

Event Property	Description
`transactionId`	UUID of the transaction; you might see the same ID 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"`.
`eventName`	Name of the audit event: `access` for this log.
`transactionId`	UUID of the transaction; you might see the same transaction 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.

Event Property	Description
`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.
`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; you might see the same transaction 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`.

Event Property	Description
`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; you might see the same transaction 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`, `MANAGED_USER` or `SOCIAL_PROVIDERS`.
`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; you might see the same transaction 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; 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.

Chapter 3

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"
- "JSON Audit Event Handler Properties"
- "CSV Audit Event Handler Properties"
- "Repository and Router Audit Event Handler Properties"
- "JMS Audit Event Handler Properties"
- "Syslog Audit Event Handler Properties"
- "Splunk Audit Event Handler Properties"

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 at which 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 exceed 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 exceed 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.
`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

Property	Description
`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 will see the original field name. For more information, see 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.
Rotation Times	`rotationTimes`	Specifies a list of times at which 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.

UI Label / Text	`audit.json` File Label	Description
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 exceed 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 exceed 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.
rotationRetentionCheckInter	`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`.
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

UI Label / Text	audit.json File Label	Description
		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.

UI Label / Text	audit.json File Label	Description
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
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

UI Label / Text	`audit.json` File Label	Description
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
Buffering	`buffering`	Disabled by default; all messages written immediately to the log

Splunk Audit Event Handler Properties

Property	Description
`useSSL`	Specifies whether IDM should connect to the Splunk instance over SSL. Boolean, true or false.
`host`	The hostname or IP address of the Splunk instance. If no hostname is specified, `localhost` is assumed.
`port`	The dedicated Splunk port for HTTP input. Default: 8088.
`buffering`	Configuration for event buffering
`maxSize`	The maximum number of events that can be buffered. Default/minimum: 10000.
`writeInterval`	The delay after which the file-writer thread is scheduled to run after encountering an empty event buffer (units of 'ms' or 's' are recommended). Default: 100 ms.
`maxBatchedEvents`	The maximum number of events per batch-write to Splunk for each Write Interval. Default: 500.
`authzToken`	The authorization token associated with the Splunk configured HTTP event collector.

Chapter 4

Configure Reports and Notifications

IDM provides a basic reporting service that enables you to generate reports on specific sets of data within a resource collection. The reporting service does not intend to replace a comprehensive data analysis platform, such as the Elastic Stack. However, this service can avoid the need for third-party data analysis tools in simple use cases.

The reporting service is accessible with a filtered query on the `openidm/report` endpoint.

The query must include an `aggregateFields` parameter. This parameter provides a comma-delimited list of name-value pairs of fields that are aggregated to generate the report. The *name* indicates the field type, and the *value* indicates the field pointer. The field type can be `TIMESTAMP` or `VALUE`. A `TIMESTAMP` field specifies a field name, a time `scale` and, optionally, a `utcOffset` in the format `+/-HHmm`. A `VALUE` field specifies a JSON pointer to any other fields to be aggregated in the report. For example:

```
TIMESTAMP=/timestamp;scale:min;utfOffset:-0700,VALUE=/response/status
```

The following example shows the full query syntax required to generate a report on a particular resource collection:

```
openidm/report/resourceCollection?_queryFilter=queryFilter&aggregateFields=TIMESTAMP=field;scale:min|hour|
day|week|month:utcOffset:offset,VALUE=field
```

Generate Audit Reports

Audit reports are intended to count similar records, usually over specified time periods. To facilitate time-based reports, audit data includes `timestamps` in ISO 8601 format (`yyyy-MM-ddTHH:mm:ss`). To aggregate the audit data for a particular time period, include these timestamps in a filtered query on the `report/audit` endpoint. You can use a UTC offset to specify different timezones.

The following example generates a report of `recon` audit events. The events are filtered to include only records with a `timestamp` value after (`gt`) October 1, 2017 and before (`lt`) October 31, 2017, both at midnight. In effect, this query generates a reconciliation report for the month of October, 2017.

The `aggregateFields` parameter determines which fields are included in the report. In the following example, the report includes the `timestamp` and `status` of each event. The `timestamp` shows the number of seconds since the Unix Epoch and the time in ISO 8601 format, with a `utcOffset` of `-0700` (which corresponds to US Pacific Daylight Time).

```
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/report/audit/recon?_queryFilter=timestamp+gt
+"2017-10-01T00:00:00.0-0700"+and+timestamp+lt+"2017-10-31T00:00:00.0-0700"&aggregateFields=TIMESTAMP=/
timestamp;scale:min;utcOffset:-0700,VALUE=/status'
{
  "result": [
    {
      "timestamp": {
        "epochSeconds": 1509361500,
        "iso8601": "2017-10-30T11:05:00.000Z"
      },
      "status": null,
      "count": 1
    },
    {
      "timestamp": {
        "epochSeconds": 1509361440,
        "iso8601": "2017-10-30T11:04:00.000Z"
      },
      "status": null,
      "count": 1
    },
    {
      "timestamp": {
        "epochSeconds": 1509361440,
        "iso8601": "2017-10-30T11:04:00.000Z"
      },
      "status": "SUCCESS",
      "count": 4
    },
    {
      "timestamp": {
        "epochSeconds": 1509361320,
        "iso8601": "2017-10-30T11:02:00.000Z"
      },
      "status": null,
      "count": 1
    },
    {
      "timestamp": {
        "epochSeconds": 1509361320,
        "iso8601": "2017-10-30T11:02:00.000Z"
      },
      "status": "SUCCESS",
      "count": 3
    },
    {
      "timestamp": {
        "epochSeconds": 1509361500,
        "iso8601": "2017-10-30T11:05:00.000Z"
      },
      "status": "SUCCESS",
      "count": 4
    }
  ],
  "resultCount": 6,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
```

```
  "remainingPagedResults": -1
}
```

You can further refine the audit report using an additional filter parameter, `postAggregationFilter`, to filter the aggregated audit results according to additional criteria. The `postAggregationFilter` parameter works in the same way as the `queryFilter` parameter.

The following example returns the same audit report generated previously but filters the aggregated results to display only those records whose `count` parameter is more than 2:

```
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/report/audit/recon?_queryFilter=timestamp+gt
+"2017-10-01T00:00:00.0-0700"+and+timestamp+lt+"2017-10-31T00:00:00.0-0700"&aggregateFields=TIMESTAMP=/
timestamp;scale:min;utcOffset:-0700,VALUE=/status&postAggregationFilter=count+gt+2'
{
  "result": [
    {
      "timestamp": {
        "epochSeconds": 1509361440,
        "iso8601": "2017-10-30T11:04:00.000Z"
      },
      "status": "SUCCESS",
      "count": 4
    },
    {
      "timestamp": {
        "epochSeconds": 1509361320,
        "iso8601": "2017-10-30T11:02:00.000Z"
      },
      "status": "SUCCESS",
      "count": 3
    },
    {
      "timestamp": {
        "epochSeconds": 1509361500,
        "iso8601": "2017-10-30T11:05:00.000Z"
      },
      "status": "SUCCESS",
      "count": 4
    }
  ],
  "resultCount": 3,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

You can sort the audit report using the `sortKeys` property. The following example runs the same query as the previous example but sorts the output according to the value of the `iso8601` field (the precise date and time of the 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/report/audit/recon?_queryFilter=timestamp+gt
+"2017-10-01T00:00:00.0-0700"+and+timestamp+lt+"2017-10-31T00:00:00.0-0700"&aggregateFields=TIMESTAMP=/
timestamp;scale:min;utcOffset:-0700,VALUE=/status&postAggregationFilter=count+gt+2&_sortKeys=timestamp/
iso8601'
{
  "result": [
    {
      "timestamp": {
        "epochSeconds": 1509361320,
        "iso8601": "2017-10-30T11:02:00.000Z"
      },
      "status": "SUCCESS",
      "count": 3
    },
    {
      "timestamp": {
        "epochSeconds": 1509361440,
        "iso8601": "2017-10-30T11:04:00.000Z"
      },
      "status": "SUCCESS",
      "count": 4
    },
    {
      "timestamp": {
        "epochSeconds": 1509361500,
        "iso8601": "2017-10-30T11:05:00.000Z"
      },
      "status": "SUCCESS",
      "count": 4
    }
  ],
  "resultCount": 3,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

Tip

The Admin UI includes an Audit Events widget that generates basic time-based reports on audit data. For more information, see "View Audit Events in the Admin UI".

Generate Reports on Managed Data

To generate a report on managed data, run a filtered query on the `report/managed` endpoint. These reports enable data analysis on areas such as:

- Number of active managed users
- Number of self-registered managed users

- Number of enabled roles

The following example generates a report on the number of managed users born in the year 1999 and aggregates those users by birth month:

```
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/report/managed/user?_queryFilter=/birthdate+gt
+"1998-12-31T23:59:59.999Z"+and+/birthdate+lt+"2000-01-01T00:00:00.000Z"&aggregateFields=TIMESTAMP=/
birthdate;scale:month;utcOffset:-0700'
{
  "result": [
    {
      "birthdate": {
        "epochSeconds": 933490800,
        "iso8601": "1999-08-01T00:00:00.0-0700"
      },
      "count": 8
    },
    {
      "birthdate": {
        "epochSeconds": 915174000,
        "iso8601": "1999-01-01T00:00:00.0-0700"
      },
      "count": 8
    },
    {
      "birthdate": {
        "epochSeconds": 917852400,
        "iso8601": "1999-02-01T00:00:00.0-0700"
      },
      "count": 10
    },
    {
      "birthdate": {
        "epochSeconds": 920271600,
        "iso8601": "1999-03-01T00:00:00.0-0700"
      },
      "count": 6
    },
    {
      "birthdate": {
        "epochSeconds": 928220400,
        "iso8601": "1999-06-01T00:00:00.0-0700"
      },
      "count": 10
    },
    {
      "birthdate": {
        "epochSeconds": 930812400,
        "iso8601": "1999-07-01T00:00:00.0-0700"
      },
      "count": 6
    },
    {
      "birthdate": {
```

```
        "epochSeconds": 936169200,
        "iso8601": "1999-09-01T00:00:00.0-0700"
      },
      "count": 7
    },
    {
      "birthdate": {
        "epochSeconds": 922950000,
        "iso8601": "1999-04-01T00:00:00.0-0700"
      },
      "count": 5
    },
    {
      "birthdate": {
        "epochSeconds": 925542000,
        "iso8601": "1999-05-01T00:00:00.0-0700"
      },
      "count": 3
    }
  ],
  "resultCount": 9,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```


The Admin UI provides a number of *count widgets* that generate reports on the following managed data:

- Number of active users (users whose account status is `active`
- Number of enabled social providers
- Number of enabled roles
- Number of configured connectors
- Number of manual user registrations

The count widgets are provided by default on the Resource Report and Business Report dashboards. Select Dashboards > Resource Report or Dashboards > Business Report to use these widgets, or add them to any other dashboard. For more information, see "Manage Dashboards" in the *Setup Guide*.

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, and include the "End User UI Notifications").

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 enabled by default, and configured in your project's `conf/notificationFactory.json` file. This file has the following structure:

```
{
    "enabled" : true,
    "threadPool" : {
        "steadyPoolThreads" : 2,
        "maxPoolThreads" : 10,
        "threadKeepAlive" : 60,
        "maxQueueSize" : 20000
    }
}
```

To disable the notification service, thereby disabling *all* notifications, set `"enabled" : false,` in `notificationFactory.json`.

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 sends notifications for password updates and profile updates. These notifications are configured in `conf/notification-passwordUpdate.json` and `conf/notification-profileUpdate.json`, respectively. You can use these default notification configuration files as the basis for setting up custom notifications.

The following excerpt from the `notification-passwordUpdate.json` file shows the structure of a notification configuration:

```
{
    "enabled" : true,
    "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.

`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`, see "*Router Configuration*" in the *Scripting Guide*.

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 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`

IDM Glossary

correlation query	A correlation query specifies an expression that matches existing entries in a source repository to one or more entries in a target repository. A correlation query might be built with a script, but it is not the same as a correlation script. For more information, see "*Correlating Source Objects With Existing Target Objects*" in the *Synchronization Guide*.
correlation script	A correlation script matches existing entries in a source repository, and returns the IDs of one or more matching entries on a target repository. While it skips the intermediate step associated with a `correlation query`, a correlation script can be relatively complex, based on the operations of the script.
entitlement	An entitlement is a collection of attributes that can be added to a user entry via roles. As such, it is a specialized type of `assignment`. A user or device with an entitlement gets access rights to specified resources. An entitlement is a property of a managed object.
JCE	Java Cryptographic Extension, which is part of the Java Cryptography Architecture, provides a framework for encryption, key generation, and digital signatures.
JSON	JavaScript Object Notation, a lightweight data interchange format based on a subset of JavaScript syntax. For more information, see the JSON site.
JSON Pointer	A JSON Pointer defines a string syntax for identifying a specific value within a JSON document. For information about JSON Pointer syntax, see the JSON Pointer RFC.

JWT	JSON Web Token. As noted in the JSON Web Token draft IETF Memo, "JSON Web Token (JWT) is a compact URL-safe means of representing claims to be transferred between two parties." For IDM, the JWT is associated with the `JWT_SESSION` authentication module.
managed object	An object that represents the identity-related data managed by IDM. Managed objects are configurable, JSON-based data structures that IDM stores in its pluggable repository. The default configuration of a managed object is that of a user, but you can define any kind of managed object, for example, groups or roles.
mapping	A policy that is defined between a source object and a target object during reconciliation or synchronization. A mapping can also define a trigger for validation, customization, filtering, and transformation of source and target objects.
OSGi	A module system and service platform for the Java programming language that implements a complete and dynamic component model. For more information, see What is OSGi? Currently, only the Apache Felix container is supported.
reconciliation	During reconciliation, comparisons are made between managed objects and objects on source or target systems. Reconciliation can result in one or more specified actions, including, but not limited to, synchronization.
resource	An external system, database, directory server, or other source of identity data to be managed and audited by the identity management system.
REST	Representational State Transfer. A software architecture style for exposing resources, using the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed.
role	IDM distinguishes between two distinct role types - provisioning roles and authorization roles. For more information, see "Managed Roles" in the *Object Modeling Guide*.
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.