Setup Guide

/ ForgeRock Identity Management 7

Latest update: 7.0.1

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

Abstract

Guide to understanding the ForgeRock® Identity Management core architecture and to getting an initial IDM deployment up and running.

Table of Contents

Overview

In this guide, you will learn about the core ForgeRock Identity Management (IDM) IDM architecture, the IDM configuration model, and how to get a basic IDM deployment up and running after installation.

Quick Start

 Architecture Learn about the IDM architecture, component modules, and services.	 Configuration Learn how IDM loads and stores its configuration, and how to change the configuration.
 Command-Line Interface Learn about the basic command-line interface (CLI) and the utilities provided to manage an IDM instance.	 Admin User Interface Learn how to configure IDM through the browser-based Admin UI.

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
Architectural Overview

This chapter introduces the IDM architecture, and describes component modules and services.

In this chapter you will learn:

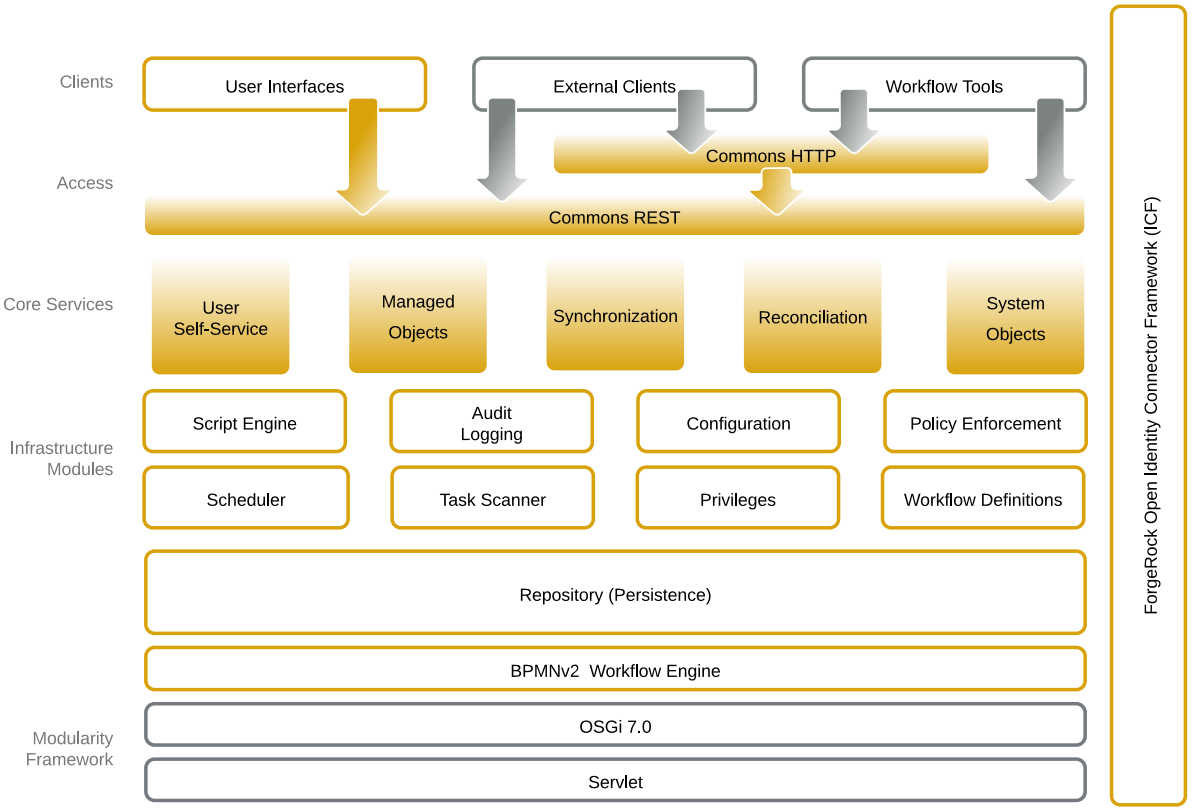
- How IDM uses the OSGi framework as a basis for its modular architecture.
- How the infrastructure modules provide the features required for IDM's core services.
- What those core services are and how they fit in to the overall architecture.
- How IDM provides access to the resources it manages.

Modular Framework

IDM implements infrastructure modules that run in an OSGi framework. It exposes core services through RESTful APIs to client applications.

The following figure provides an overview of the architecture. Specific components are described in more detail in subsequent sections of this chapter.

Modular Architecture



The IDM framework is based on OSGi:

OSGi

OSGi is a module system and service platform for the Java programming language that implements a complete and dynamic component model. For a good introduction to OSGi, see the OSGi site. IDM runs in Apache Felix, an implementation of *the OSGi Framework and Service Platform*.

Servlet

The Servlet layer provides RESTful HTTP access to the managed objects and services. IDM embeds the Jetty Servlet Container, which can be configured for either HTTP or HTTPS access.

Infrastructure Modules

The infrastructure modules provide the underlying features needed for core services:

BPMN 2.0 Workflow Engine

The embedded workflow and business process engine is based on Flowable and the Business Process Model and Notation (BPMN) 2.0 standard.

For more information, see the Workflow Guide.

Task Scanner

The task scanner performs a batch scan for a specified property, on a scheduled interval, then executes a task when the value of that property matches a specified value.

Scheduler

The scheduler supports Quartz cron triggers and simple triggers. Use the scheduler to trigger regular reconciliations, liveSync, and scripts, to collect and run reports, to trigger workflows, and to perform custom logging.

Script Engine

The script engine is a pluggable module that provides the triggers and plugin points for IDM. JavaScript and Groovy are supported.

Policy Service

An extensible policy service applies validation requirements to objects and properties, when they are created or updated.

Audit Logging

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

For more information, see "*Configure Audit Logging*" in the *Audit Guide*.

Repository

The repository provides a common abstraction for a pluggable persistence layer. IDM supports reconciliation and synchronization with several major external data stores in production, including relational databases, LDAP servers, and even flat CSV and XML files.

The repository API uses a JSON-based object model with RESTful principles consistent with the other IDM services. To facilitate testing, IDM includes an embedded instance of ForgeRock

Directory Services (DS). In production, you must use a supported repository, as described in "*Select a Repository*" in the *Installation Guide*.

Core Services

The core services are the heart of the resource-oriented unified object model and architecture:

Object Model

Artifacts handled by IDM are Java object representations of the JavaScript object model as defined by JSON. The object model supports interoperability and potential integration with many applications, services, and programming languages.

IDM can serialize and deserialize these structures to and from JSON as required. IDM also exposes a set of triggers and functions that you can define, in either JavaScript or Groovy, which can natively read and modify these JSON-based object model structures.

Managed Objects

A *managed object* is an object that represents the identity-related data managed by IDM. Managed objects are configurable, JSON-based data structures that IDM stores in its pluggable repository. The default managed object configuration includes users and roles, but you can define any kind of managed object, for example, groups or devices.

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/..."
```

System Objects

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

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

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/..."
```

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

Mappings

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

For more information, see "*Mapping Data Between Resources*" in the *Synchronization Guide*.

Reconciliation and Automatic Synchronization

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

Automatic synchronization enables creating, updating, and deleting resources from a source to a target system, either on demand or according to a schedule.

For more information, see "Types of Synchronization" in the *Synchronization Guide*.

Access Layer

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

RESTful Interfaces

IDM provides REST APIs for CRUD operations, for invoking synchronization and reconciliation, and to access several other services.

For more information, see the REST API Reference.

User Interfaces

User interfaces provide access to most of the functionality available over the REST API.

Chapter 2

Configure the Server

This chapter describes how IDM loads and stores its configuration, and how the configuration can be changed.

The configuration is defined in a combination of `.properties` files, container configuration files, and dynamic configuration objects. Most of the configuration files are stored in your project's `conf/` directory. Note that you might see files with a `.patch` extension in the `conf/` and `db/repo/conf/` directories. These files specify differences relative to the last released version of IDM and are used by the update mechanism. They do not affect your current configuration.

Configuration Objects

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

Single Instance Configuration Objects

Single instance configuration objects correspond to services that have at most one instance per installation. JSON file views of these configuration objects are named *`object-name`*`.json`.

The following list describes the single instance configuration objects:

- The `audit` configuration specifies how audit events are logged.
- The `authentication` configuration controls REST access.
- The `cluster` configuration defines how an IDM instance can be configured in a cluster.
- The `endpoint` configuration controls any custom REST endpoints.
- The `info` configuration points to script files for the customizable information service.
- The `managed` configuration defines managed objects and their schemas.
- The `policy` configuration defines the policy validation service.
- The `process-access` configuration defines access to configured workflows.
- The `repo.`*`repo-type`* configuration such as `repo.ds` or `repo.jdbc` configures the IDM repository.

- The `router` configuration specifies filters to apply for specific operations.
- The `script` configuration defines the parameters that are used when compiling, debugging, and running JavaScript and Groovy scripts.
- The `sync` configuration defines the mappings that IDM uses when it synchronizes and reconciles managed objects.
- The `ui` configuration defines the configurable aspects of the default user interfaces.
- The `workflow` configuration defines the configuration of the workflow engine.

IDM stores managed objects in the repository, and exposes them under `/openidm/managed`. System objects on external resources are exposed under `/openidm/system`.

Multiple Instance Configuration Objects

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

JSON file views of these configuration objects are named *`objectname-instancename`*`.json`, for example, `provisioner.openicf-csvfile.json.`

IDM provides the following multiple instance configuration objects:

- Multiple `schedule` configurations can run reconciliations and other tasks on different schedules.
- Multiple `provisioner.openicf` configurations correspond to connected resources.
- Multiple `servletfilter` configurations can be used for different servlet filters such as the Cross Origin and GZip filters.

Configuration Changes

When you change configuration objects, take the following points into account:

- IDM's authoritative configuration source is its repository. Although the JSON files provide a view of the configuration objects, they do not represent the authoritative source.

  Unless you have disabled file writes, IDM updates JSON files after you make configuration changes over REST. You can also edit those JSON files directly. For information on disabling file writes, see "Disabling Automatic Configuration Updates" in the *Security Guide*.

- IDM recognizes changes to JSON files when it is running. The server *must* be running when you delete configuration objects, even if you do so by editing the JSON files.

- Avoid editing configuration objects directly in the repository. Rather, edit the configuration over the REST API, or in the configuration JSON files to ensure consistent behavior and that operations are logged.
- By default, IDM stores its configuration in the repository. If you remove an IDM instance and do not specifically drop the repository, the configuration remains in effect for a new instance that uses that repository. You can disable this *persistent configuration* in your project's `conf/system.properties` file by uncommenting the following line:

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

  Disabling persistent configuration means that IDM stores its configuration in memory only.

Default REST Context

By default, IDM objects are accessible over REST at the context path `/openidm/*` where `*` indicates the remainder of the context path, for example `/openidm/managed/user`. You can change the default REST context (`/openidm`) by setting the `openidm.servlet.alias` property in your project's `resolver/boot.properties` file.

The following change to the `boot.properties` file sets the REST context to `/example`:

```
openidm.servlet.alias=/example
```

After this change, objects are accessible at the `/example` context path, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/example/managed/user?_queryFilter=true&_fields=_id"
{
  "result": [
    {
      "_id": "bjensen",
      "_rev": "0000000042b1dcd2"
    },
    {
      "_id": "scarter",
      "_rev": "000000009b54de8a"
    }
  ],
  ...
}
```

If you are using the Admin UI, you must also change the following files, so that the UI is accessible at the new context path:

- In the `/path/to/openidm/ui/admin/default/org/forgerock/openidm/ui/common/util/Constants.js` file, change the value of the `commonConstants.context` property. For example, if your new REST context is `example`, adjust that file as follows:

```
commonConstants.context = window.context || "/example";
```

- In the `/path/to/openidm/ui/admin/default/index.html` file, search for any instances of `openidm`, and change them to the new context. For example:

```
...
    xhr.open('GET', location.origin + "/example/info/uiconfig");
...
    [ location.origin + "/example" ]: uiConfig.configuration.platformSettings.adminOauthClientScopes,
...
```

Note that changing the REST context impacts the API Explorer. To use the API Explorer with the new REST context, change the `baseUrl` property in the following file:

`/path/to/openidm/ui/api/default/index.html`

Based on the change to the REST context earlier in this section, you'd set the following:

```
//base URL for accessing the OpenAPI JSON endpoint
var baseURL = '/example/';
```

HTTP I/O Buffer

When HTTP I/O requests exceed the memory limit, caching switches to a temporary file. The following lines from `boot.properties` display the default values related to buffer size:

```
# initial size of the in-memory I/O buffer for HTTP requests
#openidm.temporarystorage.initialLength.bytes=8192

# maximum size of the in-memory I/O buffer for HTTP requests
#openidm.temporarystorage.memoryLimit.bytes=65536

# maximum size of the filesystem I/O buffer for HTTP requests, for when memoryLimit is exceeded
#openidm.temporarystorage.fileLimit.bytes=1073741824

# absolute directory path of filesystem I/O buffer for HTTP requests, and uses system property
 java.io.tmpdir by default
#openidm.temporarystorage.directory=/var/tmp
```

`openidm.temporarystorage.initialLength.bytes`

Initial size of the memory buffer in bytes.

Default: 8192 bytes (8 KB). Maximum: The value of `openidm.temporarystorage.memoryLimit.bytes`.

`openidm.temporarystorage.memoryLimit.bytes`

Maximum size of the in-memory I/O buffer for HTTP requests. When the memory buffer is full, the content is transferred to a temporary file.

Default: 65536 bytes (64 KB). Maximum: 2147483647 bytes (2 GB).

`openidm.temporarystorage.fileLimit.bytes`

Maximum size of the temporary storage file. If the downloaded file is larger than this value, IDM throws the exception `HTTP 413 Payload Too Large`.

Default: 1073741824 bytes (1 GB). Maximum: 2147483647 bytes (2 GB).

`openidm.temporarystorage.directory`

The absolute directory path of the filesystem I/O buffer for HTTP requests.

Default: The value of the system property `java.io.tmpdir`.

Configure the Server Over REST

IDM exposes configuration objects under the `/openidm/config` context path.

To list the configuration on the local host, perform a GET request on `http://localhost:8080/openidm/config`.

+ *Example GET Request*

The following REST call includes excerpts of the default configuration for an IDM instance started with the `sync-with-csv` sample:

```
curl \
--request GET \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
http://localhost:8080/openidm/config
{
  "_id": "",
  "configurations": [
    {
      "_id": "router",
      "pid": "router",
      "factoryPid": null
    },
    {
      "_id": "info/login",
      "pid": "info.f01fc3ed-5871-408d-a5f0-bef00ccc4c8f",
      "factoryPid": "info"
    },
    {
      "_id": "provisioner.openicf/csvfile",
      "pid": "provisioner.openicf.9009f4a1-ea47-4227-94e6-69c345864ba7",
      "factoryPid": "provisioner.openicf"
    },
    {
      "_id": "endpoint/usernotifications",
      "pid": "endpoint.e2751afc-d169-4a23-a88e-7211d340bccb",
      "factoryPid": "endpoint"
```

```
    },
    ...
  ]
}
```

Single instance configuration objects are located under `openidm/config/object-name`.

+ *Example Audit Output*

The following example shows the `audit` configuration of the `sync-with -csv` sample.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
"http://localhost:8080/openidm/config/audit"
{
  "_id": "audit",
  "auditServiceConfig": {
    "handlerForQueries": "json",
    "availableAuditEventHandlers": [
      "org.forgerock.audit.handlers.csv.CsvAuditEventHandler",
      "org.forgerock.audit.handlers.elasticsearch.ElasticsearchAuditEventHandler",
      "org.forgerock.audit.handlers.jms.JmsAuditEventHandler",
      "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
      "org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHandler",
      "org.forgerock.openidm.audit.impl.RepositoryAuditEventHandler",
      "org.forgerock.openidm.audit.impl.RouterAuditEventHandler",
      "org.forgerock.audit.handlers.splunk.SplunkAuditEventHandler",
      "org.forgerock.audit.handlers.syslog.SyslogAuditEventHandler"
    ],
    "filterPolicies": {
      "field": {
        "excludeIf": [],
        "includeIf": []
      }
    },
    "caseInsensitiveFields": [
      "/access/http/request/headers",
      "/access/http/response/headers"
    ]
  },
  "eventHandlers": [
    {
      "class": "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
      "config": {
        "name": "json",
        "logDirectory": "&{idm.data.dir}/audit",
        "buffering": {
          "maxSize": 100000,
          "writeInterval": "100 millis"
        },
        "topics": [
          "access",
          "activity",
          "sync",
```

```
          "authentication",
          "config"
        ]
      }
    },
    {
      "class": "org.forgerock.openidm.audit.impl.RepositoryAuditEventHandler",
      "config": {
        "name": "repo",
        "enabled": false,
        "topics": [
          "access",
          "activity",
          "sync",
          "authentication",
          "config"
        ]
      }
    }
  ],
  "eventTopics": {
    "config": {
      "filter": {
        "actions": [
          "create",
          "update",
          "delete",
          "patch",
          "action"
        ]
      }
    },
    "activity": {
      "filter": {
        "actions": [
          "create",
          "update",
          "delete",
          "patch",
          "action"
        ]
      },
      "watchedFields": [],
      "passwordFields": [
        "password"
      ]
    }
  },
  "exceptionFormatter": {
    "type": "text/javascript",
    "file": "bin/defaults/script/audit/stacktraceFormatter.js"
  }
}
```

Multiple instance configuration objects are found under `openidm/config/object-name/instance-name`.

+ *Example Multiple Instance Configuration*

The following example shows the configuration for the CSV connector from the `sync-with-csv` sample.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
"http://localhost:8080/openidm/config/provisioner.openicf/csvfile"
{
  "_id": "provisioner.openicf/csvfile",
  "connectorRef": {
    "bundleName": "org.forgerock.openicf.connectors.csvfile-connector",
    "bundleVersion": "[1.5.1.4,1.6.0.0)",
    "connectorName": "org.forgerock.openicf.csvfile.CSVFileConnector"
  },
  "poolConfigOption": {
    "maxObjects": 10,
    "maxIdle": 10,
    "maxWait": 150000,
    "minEvictableIdleTimeMillis": 120000,
    "minIdle": 1
  },
  "operationTimeout": {
    "CREATE": -1,
    "VALIDATE": -1,
    "TEST": -1,
    "SCRIPT_ON_CONNECTOR": -1,
    "SCHEMA": -1,
    "DELETE": -1,
    "UPDATE": -1,
    "SYNC": -1,
    "AUTHENTICATE": -1,
    "GET": -1,
    "SCRIPT_ON_RESOURCE": -1,
    "SEARCH": -1
  },
  "configurationProperties": {
    "csvFile": "&{idm.instance.dir}/data/csvConnectorData.csv"
  },
  "resultsHandlerConfig": {
    "enableAttributesToGetSearchResultsHandler": true
  },
  "syncFailureHandler": {
    "maxRetries": 5,
    "postRetryAction": "logged-ignore"
  },
  "objectTypes": {
    "account": {
      "$schema": "http://json-schema.org/draft-03/schema",
      "id": "__ACCOUNT__",
      "type": "object",
      "nativeType": "__ACCOUNT__",
      "properties": {
        "description": {
          "type": "string",
          "nativeName": "description",
          "nativeType": "string"
```

```
        },
        "firstname": {
          "type": "string",
          "nativeName": "firstname",
          "nativeType": "string"
        },
        "email": {
          "type": "string",
          "nativeName": "email",
          "nativeType": "string"
        },
        "name": {
          "type": "string",
          "required": true,
          "nativeName": "__NAME__",
          "nativeType": "string"
        },
        "lastname": {
          "type": "string",
          "required": true,
          "nativeName": "lastname",
          "nativeType": "string"
        },
        "mobileTelephoneNumber": {
          "type": "string",
          "required": true,
          "nativeName": "mobileTelephoneNumber",
          "nativeType": "string"
        },
        "roles": {
          "type": "string",
          "required": false,
          "nativeName": "roles",
          "nativeType": "string"
        }
      }
    }
  },
  "operationOptions": {}
}
```

You can change the configuration over REST by using an HTTP PUT or HTTP PATCH request to modify the required configuration object.

+ *Example PUT Request*

The following example uses a PUT request to modify the configuration of the scheduler service, increasing the maximum number of threads that are available for the concurrent execution of scheduled tasks:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--request PUT \
--data '{
  "threadPool": {
    "threadCount": 20
  },
  "scheduler": {
    "executePersistentSchedules": {
      "$bool": "&{openidm.scheduler.execute.persistent.schedules}"
    }
  }
}' \
"http://localhost:8080/openidm/config/scheduler"
{
  "_id": "scheduler",
  "threadPool": {
    "threadCount": 20
  },
  "scheduler": {
    "executePersistentSchedules": {
      "$bool": "&{openidm.scheduler.execute.persistent.schedules}"
    }
  }
}
```

+ *Example PATCH Request*

The following example uses a PATCH request to reset the number of threads to their original value.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--request PATCH \
--data '[
  {
    "operation" : "replace",
    "field" : "/threadPool/threadCount",
    "value" : 10
  }
]' \
"http://localhost:8080/openidm/config/scheduler"
{
  "_id": "scheduler",
  "threadPool": {
    "threadCount": 10
  },
  "scheduler": {
    "executePersistentSchedules": {
      "$bool": "&{openidm.scheduler.execute.persistent.schedules}"
    }
  }
}
```

Note

Multi-version concurrency control (MVCC) is not supported for configuration objects so you do not need to specify a revision during updates to the configuration, and no revision is returned in the output.

For more information about using the REST API to update objects, see the REST API Reference.

Property Value Substitution

In an environment where you have more than one IDM instance, you might require a configuration that is similar, but not identical, across the different instances.

Property value substitution enables you to achieve the following:

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

Property value substitution uses *configuration expressions* to introduce variables into the server configuration. You set configuration expressions as the values of configuration properties. The

effective property values can be evaluated in a number of ways. For more information about property evaluation, see "Expression Evaluation and Order of Precedence".

Configuration expressions have the following characteristics:

- To distinguish them from static values, configuration expressions are preceded by an ampersand and enclosed in braces. For example: `&{openidm.port.http}`. The configuration token in the example is `openidm.port.http`. The `.` serves as the separator character.
- You can use a default value in a configuration expression by including it after a vertical bar following the token.

  For example, the following expression sets the default HTTP port value to 8080: `&{openidm.port.http|8080}`.

  With this configuration, the server attempts to substitute `openidm.port.http` with a defined configuration token. If no token definition is found, the server uses the default value, `8080`.
- A configuration property can include a mix of static values and expressions.

  For example, suppose `hostname` is set to `ds`. Then `&{hostname}.example.com` evaluates to `ds.example.com`.
- Configuration token evaluation is recursive.

  For example, suppose `port` is set to `&{port.prefix}389`, and `port.prefix` is set to `2`. Then `&{port}` evaluates to `2389`.

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

Important

Property substitution is *not* available for any configuration not processed by the IDM backend, such as `ui-themeconfig` or any user-supplied configuration.

Expression Evaluation and Order of Precedence

At server startup, expression resolvers evaluate property values to determine the effective configuration. You must define expression values before you start the IDM server that uses them.

When configuration tokens are resolved, the result is always a string. However, you can *coerce* the output type of the evaluated token to match the type that is required by the property. Ultimately, the expression must return the appropriate data type for the configuration property. For example, the `port` property takes an integer. If you set it using an expression, the result of the evaluated expression must be an integer. If the type is wrong, the server fails to start due to a syntax error. For more information about data type coercion, see "Transforming Data Types".

Expression resolvers can obtain values from the following sources:

1. **Environment variables**

   You set an environment variable to hold the property value.

   For example: `export OPENIDM_PORT_HTTP=8080`

   The environment variable name must be composed of uppercase characters and underscores. The name maps to the expression token as follows:

   - Uppercase characters are converted to lowercase.
   - Underscores, `_`, are replaced with `.` characters.

   In other words, the value of `OPENIDM_PORT_HTTP` replaces `&{openidm.port.http}` in the server configuration.

2. **Java system properties**

   You set a Java system property to hold the value.

   Java system property names must match expression tokens exactly. In other words, the value of the `openidm.repo.port` system property replaces `&{openidm.repo.port}` in the server configuration.

   Java system properties can be set in a number of ways. One way of setting system properties for IDM servers is to pass them through the `OPENIDM_OPTS` environment variable.

   For example: `export OPENIDM_OPTS="-Dopenidm.repo.port=3306"`

   System properties can also be declared in your project's `conf/system.properties`.

   This example uses property value substitution with a standard system property. The example modifies the audit configuration, changing the `audit.json` file to redirect JSON audit logs to the user's home directory. The `user.home` property is a default Java System property:

```
"eventHandlers" : [
    {
        "class" : "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
        "config" : {
            "name" : "json",
            "logDirectory" : "&{user.home}/audit",
            ...
        }
    },
    ...
]
```

3. **Expression files**

   You set a key in a `.json` or `.properties` file to hold the value. To use an expression file, set the `IDM_ENVCONFIG_DIRS` environment variable, or the `idm.envconfig.dirs` Java system property as described below. By default, IDM sets `idm.envconfig.dirs` to `&{idm.install.dir}/resolver/`.

The default property resolver file in IDM is `resolver/boot.properties` but you can specify additional files that might hold property values.

Keys in `.properties` files must match expression tokens exactly. In other words, the value of the `openidm.repo.port` key replaces `&{openidm.repo.port}` in the server configuration.

The following example expression properties file sets the repository port:

```
openidm.repo.port=1389
```

JSON expression files can contain nested objects.

JSON field names map to expression tokens as follows:

- The JSON path name matches the expression token.
- The `.` character serves as the JSON path separator character.

The following example JSON expression file uses property value substitution to set the host in the LDAP connector configuration:

```
{
    "openidm" : {
        "provisioner" : {
            "ldap" : {
                "host" : "ds.example.com"
            }
        }
    }
   }
 }
```

To substitute this value in the configuration, the LDAP provisioner file would include the following:

```
{
    ...
    "configurationProperties" : {
        "host" : &{openidm.provisioner.ldap.host|localhost},
        ...
    }
}
```

If the server does not find a configuration token for the host name, it substitutes the default (`localhost`).

To use expression files, set the environment variable, `IDM_ENVCONFIG_DIRS`, or the Java system property, `idm.envconfig.dirs`, to a comma-separated list of the directories containing the expression files.

When reading these files, the server browses the directories in the order specified. It reads all the files with `.json` and `.properties` extensions, and attempts to use them to evaluate expression tokens.

For example, if you define `idm.envconfig.dirs=/directory1,/directory2` and a configuration token is defined in both `directory1` and `directory2`, the resolved value will be the value defined in `directory1`. If the configuration token is defined only in `directory2`, the resolved value will be the value defined in `directory2`.

Note the following constraints when using expression files:

- Although IDM scans the directories in a specified order, within a directory IDM scans the files in a nondeterministic order.
- IDM does not scan subdirectories.
- Do *not* define the same configuration token more than once in a file.

  If you define the same property twice in the same file, one definition will be used and the other will be ignored. The server will not throw an error, but because files are scanned in a nondeterministic order, you have no way of knowing which value will be used.
- You cannot define the same configuration token in more than one file in a single directory. The server generates an error in this case.

  Important

  This constraint implies that you cannot have backup `.properties` and `.json` files, in a single directory if they define the same tokens.
- If the same token occurs once in several files that are located in different directories, IDM uses the first value that is read.

4. **Framework configuration properties**

   You can use the `conf/config.properties` file to override values used by the OSGI framework.

5. **Configuration files**

   All the properties declared in the `.json` files in your project's `conf/` directory.

Evaluation Order of Precedence

The following list displays the order of precedence, from greatest to least:

1. Environment variables override system properties, default token settings, and settings in expression files.
2. System properties override default token settings, and any settings in expression files.
3. Default token settings.

4. If `IDM_ENVCONFIG_DIRS` or `idm.envconfig.dirs` is set, the server uses the settings found in expression files.
5. Framework configuration properties.
6. Hardcoded property values.
7. Properties passed to the startup script with options such as: `-P`, `-w`, and `-s`. in the *Installation Guide*

Transforming Data Types

When configuration tokens are resolved, the result is always a string. However, you can transform or *coerce* the output type of the evaluated token to match the type that is required by the property.

You transform a property's data type by setting `$type` before the property value.

The following coercion types are supported:

- array (`$array`)
- boolean (`$bool`)
- decodeBase64 (`$base64:decode`)

  Transforms a base64-encoded string into a decoded string.
- encodeBase64 (`$base64:encode`)

  Transforms a string into a base64-encoded string.
- integer (`$int`)
- list (`$list`)
- number (`$number`)

  This type can coerce integers, doubles, longs, and floats.
- object (`$object`)

  This type can coerce a JSON object such as an encrypted password.

+ *Type Coercion to Integer*

  This example JSON expression file sets the value of the port in the LDAP connector configuration:

```
{
    "openidm" : {
        "provisioner" : {
            "ldap" : {
                "port" : 6389
            }
        }
    }
}
```

When the expression is evaluated, the port is evaluated as a `string` value, (which would cause an error). To coerce the port value to an integer, substitute the value in the LDAP provisioner file as follows:

```
{
    ...
    "configurationProperties" : {
        "port" : {
            "$int" : "&{openidm.provisioner.ldap.port|1389}",
        },
        ...
    }
}
```

With this configuration, the server evaluates the LDAP port property to the integer `6389`. If the server does not find a configuration token for the port, it substitutes the default (`1389`).

+ *Type Coercion to Array*

This example JSON expression file sets a value for the failover servers in an LDAP connector configuration:

```
"openidm" : {
    "provisioner" : {
        "ldap" : {
            "failover" : ["ldap://host1.domain.com:1389", "ldap://host2.domain.com:1389"]
        }
    }
}
```

When the expression is evaluated, the URLs would be evaluated as a single `string`. To coerce the value to an array, substitute the value in the LDAP provisioner file as follows:

```
{
    ...
    "configurationProperties" : {
        "failover" : {
            "$array":"&{openidm.provisioner.ldap.failover}",
        },
        ...
    }
}
```

Note

If you set the failover URLs in a `.properties` file, instead of in a `.json` file, you *must* escape the JSON object. This example sets the failover servers array in the `boot.properties` file:

```
openidm.provisioner.ldap.failover=[\"ldap://host1.domain.com:1389\",\"ldap://
host2.domain.com:1389\"]
```

+ *Type Coercion From List to Array*

The `$list` function is similar to `$array`, but lets you specify values in a `.properties` file as a list of strings, separated by a comma (`,`).

For example, you could list the LDAP failover servers in `boot.properties` as follows:

```
openidm.provisioner.ldap.failover=ldap://host1.domain.com:1389,ldap://host2.domain.com:1389
```

To coerce the value to an array, your property definition in the LDAP provisioner file would be:

```
{
    ...
    "configurationProperties" : {
        "failover" : {
            "$list":"&{openidm.provisioner.ldap.failover}",
        },
        ...
    }
}
```

This configuration would be converted to:

```
"openidm" : {
    "provisioner" : {
        "ldap" : {
            "failover" : ["ldap://host1.domain.com:1389", "ldap://host2.domain.com:1389"]
        }
    }
}
```

Configuration Property Value Storage

The values of configuration properties that are set explicitly (in `conf/*.json` files) are stored in the repository. You can manage these configuration objects over REST or by using the JSON files themselves.

Properties that use value substitution are stored in the repository as *variables*. You store the *value* of each variable in `*.properties` files. You can use different `*.properties` files to change the configuration for multiple nodes in a cluster.

The following table shows how specific configuration properties can be set:

Configuration Property Variables

Variable	Description	Environment Variables	System Variables	`boot.properties`
`idm.install.dir`	Directory of files from unpacked IDM binary	X	X	X
`idm.data.dir`	Working location directory	X	X	X
`idm.instance.dir`	Project directory with IDM configuration files	X	X	X
`idm.envconfig.dirs`	Directory with environment files, including `boot.properties`	X	X	

You can access configuration properties in scripts using `identityServer.getProperty()`. For more information, see "The `identityServer` Variable" in the *Scripting Guide*.

Limitations of Property Value Substitution

Property value substitution is limited in the following areas:

+ *In the Admin UI*

  Support for property value substitution in the Admin UI is limited to the following categories:

  - String substitution, where `&{some.property|DefaultValue}`
  - Number and integer substitution, including:
    - `"$number" : "&{openidm.port|1234}"`
    - `"$int" : "&{openidm.port|5678}"`
  - Base64 substitution, such as: `"$base64:decode" : "&{openidm.felix.web.console.password|YWRtaW4=}"`
  - Cryptographic substitution, where for passwords and client secrets, IDM substitutes `"********"` for `$crypto`

+ *In Connector Configurations*

You cannot use property substitution for connector reference (`connectorRef`) properties. For example, the following configuration is *not* valid:

```
"connectorRef" : {
    "connectorName" : "&{connectorName}",
    "bundleName" : "org.forgerock.openicf.connectors.ldap-connector",
    "bundleVersion" : "&{LDAP.BundleVersion}"
    ...
}
```

The `connectorName` must be the precise string from the connector configuration. To specify multiple connector version numbers, use a range of versions. For example:

```
"connectorRef" : {
    "connectorName" : "org.identityconnectors.ldap.LdapConnector",
    "bundleName" : "org.forgerock.openicf.connectors.ldap-connector",
    "bundleVersion" : "[1.5.0.0,2.0.0.0)",
    ...
}
```

HTTP Clients

Several IDM modules, such as the external REST service and identity provider service, need to make HTTP(S) requests to external systems.

HTTP client settings can be configured through any expression resolver (in `resolver/boot.properties`, environment variables, or Java system properties). Configuration for specific clients can be set in that client's JSON configuration file. For example `conf/external.rest.json` configures the external REST service and properties set there override the expression resolvers. For more information on property resolution, see "Expression Evaluation and Order of Precedence".

You can set the following properties for HTTP clients:

`openidm.http.client.sslAlgorithm`

The cipher to be used when making SSL/TLS connections, for example, `AES`, `CBC`, or `PKCS5Padding`. Defaults to the system SSL algorithm.

`openidm.http.client.socketTimeout`

The TCP socket timeout, in seconds, when waiting for HTTP responses. The default timeout is 10 seconds.

`openidm.http.client.connectionTimeout`

The TCP connection timeout for new HTTP connections, in seconds. The default timeout is 10 seconds.

`openidm.http.client.reuseConnections` **(true or false)**

Specifies whether HTTP connections should be kept alive and reused for additional requests. By default, connections will be reused if possible.

`openidm.http.client.retryRequests` **(true or false)**

Specifies whether requests should be retried if a failure is detected. By default requests will be retried.

`openidm.http.client.maxConnections` **(integer)**

The maximum number of connections that should be pooled by the HTTP client. At most `64` connections will be pooled by default.

`openidm.http.client.hostnameVerifier` **(string)**

Specifies whether the client should check that the hostname to which it has connected is allowed by the certificate that is presented by the server.

The property can take the following values:

- `STRICT` - hostnames are validated
- `ALLOW_ALL` - the external REST service does not attempt to match the URL hostname to the SSL certificate Common Name, as part of its validation process

If you do not set this property, the behavior is to validate hostnames (the equivalent of setting `"hostnameVerifier": "STRICT"`). In production environments, you *should* set this property to `STRICT`.

`openidm.http.client.proxy.uri`

Specifies that the client should make its HTTP(S) requests through the specified proxy server.

`openidm.http.client.proxy.userName`

The username of the account for the specified proxy.

`openidm.http.client.proxy.password`

The password of the account for the specified proxy.

`openidm.http.client.proxy.useSystem` **(true or false)**

If `true`, specifies a system-wide proxy with the JVM system properties, `http.proxyHost`, `http.proxyPort`, and (optionally) `http.nonProxyHosts`.

If `openidm.http.client.proxy.uri` is set, and not empty, that setting overrides the system proxy setting.

Chapter 3
Command-Line Interface

This chapter describes the basic command-line interface (CLI). The CLI includes a number of utilities for managing an IDM instance.

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

```
/path/to/openidm/cli.sh
Using boot properties at /path/to/openidm/resolver/boot.properties
openidm# encrypt ....
```

or

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

The command-line utilities run with the security properties defined in your project's `conf/secrets.json in the Security Guide` file.

If you run the **cli.sh** command by itself, it opens an IDM-specific shell prompt:

```
openidm#
```

The startup and shutdown scripts are not discussed in this chapter. For information about these scripts, see "*Configuration and Monitoring*" in the *Installation Guide*.

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

For a list of subcommands available from the `openidm#` prompt, run the **cli.sh help** command. The **help** and **exit** options shown below are self-explanatory. The other subcommands are explained in the subsections that follow:

```
local:secureHash   Hash the input string.
local:keytool      Export or import a SecretKeyEntry. The Java Keytool does not allow for exporting or
 importing SecretKeyEntries.
local:encrypt      Encrypt the input string.
local:validate     Validates all json configuration files in the configuration (default: /conf) folder.
basic:help   Displays available commands.
basic:exit   Exit from the console.
remote:configureconnector   Generate connector configuration.
remote:configexport         Exports all configurations.
remote:update               Update the system with the provided update file.
remote:configimport         Imports the configuration set from local file/directory.
```

The following options are common to the **configexport**, **configimport**, and **configureconnector** subcommands:

-u or --user USER[:PASSWORD]

Allows you to specify the server user and password. Specifying a username is mandatory. If you do not specify a username, the following error is output to the OSGi console: `Remote operation failed: Unauthorized`. If you do not specify a password, you are prompted for one. This option is used by all three subcommands.

--url URL

The URL of the REST service. The default URL is `http://localhost:8080/openidm/`. This can be used to import configuration files from a remote running IDM instance. This option is used by all three subcommands.

-P or --port PORT

The port number associated with the REST service. If specified, this option overrides any port number specified with the **--url** option. The default port is `8080`. This option is used by all three subcommands.

configexport

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

OpenIDM must be running when you execute this command.

Usage is as follows:

```
./cli.sh configexport --user username:password export-location
```

For example:

```
./cli.sh configexport --user openidm-admin:openidm-admin /tmp/conf
```

On Windows systems, the *export-location* must be provided in quotation marks, for example:

```
C:\openidm\cli.bat configexport --user openidm-admin:openidm-admin "C:\temp\openidm"
```

Configuration objects are exported as `.json` files to the specified directory. The command creates the directory if needed. Configuration files that are present in this directory are renamed as backup files, with a timestamp, for example, `audit.json.2014-02-19T12-00-28.bkp`, and are not overwritten. The following configuration objects are exported:

- The internal repository table configuration (`repo.ds.json` or `repo.jdbc.json`) and the datasource connection configuration, for JDBC repositories (`datasource.jdbc-default.json`)

- The script configuration (`script.json`)
- The log configuration (`audit.json`)
- The authentication configuration (`authentication.json`)
- The cluster configuration (`cluster.json`)
- The configuration of a connected SMTP email server (`external.email.json)`
- Custom configuration information (`info-*name*.json`)
- The managed object configuration (`managed.json`)
- The connector configuration (`provisioner.openicf-*.json`)
- The router service configuration (`router.json`)
- The scheduler service configuration (`scheduler.json`)
- Any configured schedules (`schedule-*.json`)
- Standard security questions (`selfservice.kba.json)`
- The mapping configuration
- If workflows are defined, the configuration of the workflow engine (`workflow.json`) and the workflow access configuration (`process-access.json`)
- Any configuration files related to the user interface (`ui-*.json`)
- The configuration of any custom endpoints (`endpoint-*.json`)
- The configuration of servlet filters (`servletfilter-*.json`)
- The policy configuration (`policy.json`)

configimport

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

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

The subcommand takes the following options:

`-r`, `--replaceall`, `--replaceAll`

Replaces the entire list of configuration files with the files in the specified import location.

Note that this option wipes out the existing configuration and replaces it with the configuration in the *import-location*. Objects in the existing configuration that are not present in the *import-location* are deleted.

`--retries` **(integer)**

This option specifies the number of times the command should attempt to update the configuration if the server is not ready.

Default value : 10

`--retryDelay` **(integer)**

This option specifies the delay (in milliseconds) between configuration update retries if the server is not ready.

Default value : 500

Usage is as follows:

```
./cli.sh configimport --user username:password [--replaceAll] [--retries integer] [--
retryDelay integer] import-location
```

For example:

```
./cli.sh configimport --user openidm-admin:openidm-admin --retries 5 --retryDelay 250 --replaceAll /tmp/
conf
```

On Windows systems, the *import-location* must be provided in quotation marks, for example:

```
C:\openidm\cli.bat configimport --user openidm-admin:openidm-admin --replaceAll "C:\temp\openidm"
```

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

configureconnector

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

Usage is as follows:

```
./cli.sh configureconnector --user username:password --name connector-name
```

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

```
./cli.sh configureconnector --user openidm-admin:openidm-admin --name myCsvConnector
Executing ./cli.sh...
Starting shell in /path/to/openidm
Mar 26, 2020 06:08:52 PM org.forgerock.openidm.core.FilePropertyAccessor loadProps
0. SSH Connector version 1.5.6.0
1. ServiceNow Connector version 1.5.5.0
2. Scripted SQL Connector version 1.5.5.0
3. Scripted REST Connector version 1.5.6.0
4. Scim Connector version 1.5.8.0
5. Salesforce Connector version 1.5.7.0
6. MongoDB Connector version 1.5.6.0
7. Marketo Connector version 1.5.4.0
8. LDAP Connector version 1.5.5.0
9. Kerberos Connector version 1.5.5.0
10. Scripted Poolable Groovy Connector version 1.5.7.0
11. Scripted Groovy Connector version 1.5.7.0
12. GoogleApps Connector version 1.4.9.0
13. Database Table Connector version 1.5.3.0
14. CSV File Connector version 1.5.7.0
15. Adobe Marketing Cloud Connector version 1.5.4.0
16. Exit
Select [0..17]:   14
Edit the configuration file and run the command again. The configuration was saved to
  /path/to/openidm/temp/provisioner.openicf-myCsvConnector.json
```

The basic configuration is saved in a file named `/openidm/temp/provisioner.openicf-connector-name.json`. Edit at least the `configurationProperties` parameter in this file to complete the connector configuration. For example, for a CSV connector:

```
"configurationProperties" : {
    "headerPassword" : "password",
    "csvFile" : "&{idm.instance.dir}/data/csvConnectorData.csv",
    "newlineString" : "\n",
    "headerUid" : "uid",
    "quoteCharacter" : "\"",
    "fieldDelimiter" : ",",
    "syncFileRetentionCount" : 3
}
```

For more information about the connector configuration properties, see "*Configure Connectors*" in the *Connectors Guide*.

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

```
./cli.sh configureconnector --user openidm-admin:openidm-admin --name myCsvConnector
Executing ./cli.sh...
Starting shell in /path/to/openidm
Using boot properties at /path/to/openidm/resolver/boot.properties
Configuration was found and read from: /path/to/openidm/temp/provisioner.openicf-myCsvConnector.json
```

You can now copy the new `provisioner.openicf-myCsvConnector.json` file to your project's `conf/` subdirectory.

You can also configure connectors over the REST interface, or through the Admin UI. For more information, see "*Configure Connectors*" in the *Connectors Guide*.

encrypt

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

Usage is as follows:

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

If you do not enter the string as part of the command, the command prompts for the string to be encrypted. If you enter the string as part of the command, special characters such as quotation marks, must be escaped.

-j or --json

Indicates that the string to be encrypted is a JSON object, and validates the object. If the object is malformed JSON and you use the `-j` option, the command throws an error. It is easier to input JSON objects in interactive mode. If you input the JSON object on the command-line, the object must be surrounded by quotes, and any special characters, including curly braces, must be escaped.

For example:

```
./cli.sh encrypt \
--json '\{\"password\":\"myPassw0rd\"\}'
```

The following example encrypts a normal string value:

```
./cli.sh encrypt \
mypassword

Executing ./cli.sh...
Starting shell in /path/to/openidm
-----BEGIN ENCRYPTED VALUE-----
{
  "$crypto" : {
    "type" : "x-simple-encryption",
    "value" : {
      "cipher" : "AES/CBC/PKCS5Padding",
      "stableId" : "openidm-sym-default",
      "salt" : "vdz6bUztiT6QsExNrZQAEA==",
      "data" : "RgMLRbX0guxF80nwrtaZkkoFFGqSQdNWF7Ve0zS+N1I=",
      "keySize" : 16,
      "purpose" : "idm.config.encryption",
      "iv" : "R9w1TcWfbd9FPmOjfvMhZQ==",
      "mac" : "9pXtSKAt9+dO3Mu0NlrJsQ=="
    }
  }
}
------END ENCRYPTED VALUE------
```

The following example prompts for a JSON object to be encrypted:

```
./cli.sh encrypt --json
Using boot properties at /path/to/openidm/resolver/boot.properties
Enter the Json value

> Press ctrl-D to finish input
Start data input:
{"password":"myPassw0rd"}
^D
-----BEGIN ENCRYPTED VALUE-----
{
  "$crypto" : {
    "type" : "x-simple-encryption",
    "value" : {
      "cipher" : "AES/CBC/PKCS5Padding",
      "stableId" : "openidm-sym-default",
      "salt" : "vdz6bUztiT6QsExNrZQAEA==",
      "data" : "RgMLRbX0guxF80nwrtaZkkoFFGqSQdNWF7Ve0zS+N1I=",
      "keySize" : 16,
      "purpose" : "idm.config.encryption",
      "iv" : "R9w1TcWfbd9FPmOjfvMhZQ==",
      "mac" : "9pXtSKAt9+dO3Mu0NlrJsQ=="
    }
  }
}
------END ENCRYPTED VALUE------
```

secureHash

The **secureHash** subcommand hashes an input string, or JSON object, using the specified hash algorithm configuration. Use this subcommand to hash password values, or other sensitive data, to be stored in the repository. The hashed value is output to standard output and provides details of the algorithm configuration that was used to hash the data.

Usage is as follows:

```
/path/to/openidm/cli.sh secureHash --algorithm --config [--json] string
```

-a or --algorithm

Specifies the hash algorithm to use.

-c or --config

Lets you provide additional hashing configuration options, as a JSON object. For a list of supported hash algorithms and their configuration, see "Encoding Attribute Values by Using Salted Hash Algorithms" in the *Security Guide*.

-j or --json

Indicates that the string to be encrypted is a JSON object, and validates the object. If the object is malformed JSON and you use the `-j` option, the command throws an error. It is easier to input

JSON objects in interactive mode. If you input the JSON object on the command-line, the object must be surrounded by quotes, and any special characters, including curly braces, must be escaped.

For example:

```
/path/to/openidm/cli.sh secureHash \
--algorithm SHA-384 \
--json '\{\"password\":\"myPassw0rd\"\}'
```

If you do not enter the string as part of the command, the command prompts for the string to be hashed. If you enter the string as part of the command, any special characters, for example quotation marks, must be escaped.

The following example hashes a password value (`mypassword`) using the `PBKDF2` algorithm:

```
/path/to/openidm/cli.sh secureHash \
--algorithm PBKDF2 \
--config '{\"hashLength\":16\,\"saltLength\":16\,\"iterations\":20000\,\"hmac\":\"SHA3-256\"}' \
"mypassword"
Executing ./cli.sh...
Starting shell in /path/to/openidm
...
-----BEGIN HASHED VALUE-----
{
  "$crypto" : {
    "value" : {
      "algorithm" : "PBKDF2",
      "data" : "9/1IIaAVxAMFdCzlMGtkXMmotKqBafIdx2KFUeKHX0k=",
      "config" : {
        "hashLength" : 16,
        "saltLength" : 16,
        "iterations" : 20000,
        "hmac" : "SHA3-256"
      }
    },
    "type" : "salted-hash"
  }
}
------END HASHED VALUE------
```

The following example prompts for a JSON object to be hashed:

```
/path/to/openidm/cli.sh secureHash --algorithm SHA-384 --json
Executing ./cli.sh...
Executing ./cli.sh...
Starting shell in /path/to/openidm
Nov 14, 2017 1:24:26 PM org.forgerock.openidm.core.FilePropertyAccessor loadProps
INFO: Using properties at /path/to/openidm/resolver/boot.properties
Enter the Json value

> Press ctrl-D to finish input
Start data input:
{"password":"myPassw0rd"}
^D
-----BEGIN HASHED VALUE-----
{
  "$crypto" : {
    "value" : {
      "algorithm" : "SHA-384",
      "data" : "7Caabx7d+vOZ7d3VMwdQObQJdTQ3uGOItsX5AwR4ViygUfARR/XuxRIBQt1LRq58ZOQXFwuw+3rvzK7Kld8pSg=="
    },
    "type" : "salted-hash"
  }
}
------END HASHED VALUE------
```

keytool

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

The Java **keytool** command enables you to export and import public keys and certificates, but not secret or symmetric keys. The IDM **keytool** subcommand provides this functionality.

Usage is as follows:

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

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

```
./cli.sh keytool --export openidm-sym-default
Executing ./cli.sh...
Starting shell in /home/idm/openidm
Use KeyStore from: /openidm/security/keystore.jceks
Please enter the password:
[OK] Secret key entry with algorithm AES
AES:606d80ae316be58e94439f91ad8ce1c0
```

The default keystore password is `changeit`. For security reasons, you *must* change this password in a production environment. For information about changing the keystore password, see "Changing the Default Keystore Password" in the *Security Guide*.

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

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

If a secret key with that name already exists, IDM returns the following error:

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

validate

The **validate** subcommand validates all .json configuration files in your project's `conf/` directory.

Usage is as follows:

```
./cli.sh validate
Executing ./cli.sh
Starting shell in /path/to/openidm
Using boot properties at /path/to/openidm/resolver/boot.properties
.....................................................................
[Validating] Load JSON configuration files from:
[Validating]  /path/to/openidm/conf
[Validating] audit.json ................................ SUCCESS
[Validating] authentication.json ....................... SUCCESS
    ...
[Validating] sync.json ................................. SUCCESS
[Validating] ui-configuration.json ..................... SUCCESS
[Validating] ui-countries.json ......................... SUCCESS
[Validating] workflow.json ............................. SUCCESS
```

update

The **update** subcommand supports updates to patch bundle releases. Patch bundle releases contain a collection of fixes and minor RFEs that have been grouped together.

Note

You cannot use the **update** command to upgrade to new major versions. For information on upgrading to IDM 7, see the Upgrade Guide.

Chapter 4
Admin UI

IDM provides two customizable, browser-based user interfaces: an Administrative User Interface (Admin UI) and an End User interface (End User UI).

The Admin UI provides a graphical interface for most aspects of the IDM configuration. If IDM is installed on the local system, access the Admin UI at `https://localhost:8443/admin`. The Admin UI lets you configure and manage users and roles, set up synchronization between resources, configure connectors, and more.

The End User UI provides role-based access to tasks based on BPMN2 workflows, and allows users to manage certain aspects of their own accounts, including configurable self-service registration. End users access the UI at a URL you specify. As an administrator, if IDM is installed on the local system, you can access the End User UI at `https://localhost:8443/`. All users, including `openidm-admin`, can change their password through the End User UI. The End User UI is described in "Self-Service End User UI" in the *Self-Service Reference*.

Manage Dashboards

Dashboards let you make shortcuts to frequently-required tasks. The Quick Start dashboard is displayed by default when you log in to the Admin UI. You can create additional dashboards, or add and remove widgets from the existing dashboards.

To display all configured dashboards, select Dashboards > Manage Dashboards. The following dashboards are provided by default:

Quick Start Dashboard

Quick Start cards support one-click access to common administrative tasks:

- `Add Connector`: Configure connections to external resources.
- `Create Mapping`: Configure mappings to synchronize objects between resources (see "*Mapping Data Between Resources*" in the *Synchronization Guide*).
- `Manage Roles`: Set up provisioning or authorization roles (see "Managed Roles" in the *Object Modeling Guide*).
- `Add Device`: Configure managed objects, including users, groups, roles, and devices.
- `Configure Registration`: Configure user Self-Registration. You can set up the End User UI login screen, with a link that allows new users to start a verified account registration process.

- `Configure Password Reset`: Configure user self-service Password Reset. You can configure the ability for users to reset forgotten passwords. For more information, see "Self-Service End User UI" in the *Self-Service Reference*.
- `Manage Users`: Lets you manage users in the *Object Modeling Guide* in the repository.
- `Configure System Preferences`: Configure the following aspects of the server:
  - Audit.
  - End User UI path, as described in "Change the End User UI Path" in the *Self-Service Reference*.
  - Privacy & Consent, as described in "Configure Privacy and Consent" in the *Self-Service Reference*.
  - Workflows.

System Monitoring Dashboard

- Audit Events in the *Audit Guide* include information on audit data, organised by date.
- Last Reconciliation includes data from the most recent reconciliation between data stores.

Resource Report

- The Resources Report includes widgets that show the number of active users, configured roles, and active connectors.
- The Resources widget shows all configured connectors, mappings, and managed object types.

Business Report

- The Business Report includes widgets related to login and registration activity.

Caution

If your browser uses an AdBlock extension, it might inadvertently block some UI functionality, particularly if your configuration includes strings such as `ad`. For example, a connection to an Active Directory server might be configured at the endpoint `system/ad`. To avoid problems related to blocked UI functionality, either remove the AdBlock extension, or set up a suitable safelist to ensure that none of the targeted endpoints are blocked.

You can set up additional dashboards for customized views of the Admin UI, and you can embed external dashboards such as Kibana or Grafana, or other monitoring boards.

+ *Create a New Dashboard*

To create a new dashboard, select Dashboards > New Dashboard. Enter a dashboard name and select whether this dashboard should be the default board that is displayed when you load the Admin UI.

For a customized view of the Admin UI, select Widgets as the Dashboard Type, then select Create Dashboard and add the widgets that you want exposed in that view.

You can also customize the view by starting with an existing dashboard. In the upper-right corner of the UI, next to the Add Widget button, click the vertical ellipses (⋮), then select Rename or Duplicate.

+ *Embed an External Dashboard*

1. Create a dashboard and Select Embedded URL as the Dashboard Type.
2. Enter the URL of the dashboard that you want to embed, and select Create Dashboard.

+ *Add and Move Widgets*

To add a widget to a dashboard, click Add Widget and select the widget type. Widgets are grouped in logical categories, so scroll down to the category that fits the widget you want to add.

To change the position of a widget on a dashboard, click and drag the move icon (✣).

To add a new Quick Start widget, select the vertical ellipsis (⋮) icon in the upper right corner of the widget, and click Settings.

To embed an Admin UI sub-widget in the Quick Start widget, specify the destination URL. If you are linking to a specific page in the Admin UI, the destination URL can be the part of Admin UI address. For example, to create a quick start link to the Audit configuration tab, at `https://localhost:8443/admin/#settings/audit/`, enter `#settings/audit` in the destination URL text box.

Any changes that you make to the dashboards are written to your project's `conf/ui-dashboard.json` file. That file has the following properties:

Admin UI Widget Properties in `ui-dashboard.json`

Property	Options	Description
`name`	User entry	Dashboard name
`isDefault`	`true` or `false`	Default dashboard; can set one default
`widgets`	Different options for `type`	Code blocks that define a widget
`type`	`lastRecon`, `resourceList`, `quickStart`, `frame`, `userRelationship`	Widget name
`size`	`x-small`, `small`, `medium`, or `large`	Width of widget, based on a 12-column grid system, where x-small=4, small=6, medium=8, and

Property	Options	Description
		large=12; for more information, see Bootstrap CSS
`height`	Height, in units such as `cm`, `mm`, `px`, and `in`	Height; applies only to Embed Web Page widget
`frameUrl`	URL	Web page to embed; applies only to Embed Web Page widget
`title`	User entry	Label shown in the UI; applies only to Embed Web Page widget
`barchart`	`true` or `false`	Reconciliation bar chart; applies only to Last Reconciliation widget

+ *Default Admin UI Widgets*

The following tables display an alphabetical list of the available widgets:

Admin UI Reporting Widgets

Name	Description
Audit Events	Graphical display of audit events.
Count Widget	A reporting widget that provides an instant display of the number of specific objects; for example, active managed users, and enabled social providers.
Dropwizard Table With Graph	Does not appear in the list of widgets unless metrics are active.
Graph Widget	Provides a graphical view of a specific managed resource; for example, managed users, based on some metric.
Last Reconciliation	Shows statistics from the most recent reconciliation, shown on the System Monitoring dashboard.
New Registrations	The number of users that have self-registered that week. To display data using this widget, you must enable user self-registration.
Password Resets	The number of password resets that week. To display data using this widget, you must enable password reset.
Resources	Connectors, mappings, managed objects; shown in Administration dashboard
Sign-Ins	The number of managed users that have signed in to the service that week.

Social Widgets

Name	Description
Daily Social Logins	Graphical display of logins through social identity providers.
Social Registration (year)	Graphical display of social registrations over the past year.

System Status Widgets

Name	Description
Cluster Node Status	Lists the instances in a cluster, with their status.

Utility Widgets

Name	Description
Embed Web Page	Supports embedding of external content. Web pages that you embed using this applet must support the correct `x-frame-options`. For example, Google has a Maps Embed API.
Identity Relationships	Graphical display of relationships between identities.
Managed Objects Relationship Diagram	Graphical diagram with connections between managed object properties; also see "View the Relationship Configuration in the UI" in the *Object Modeling Guide*
Quick Start	Links to common tasks; shown in Administration dashboard

Customize the Admin UI

This section shows you how to customize the Admin UI for your deployment.

The default Admin UI configuration files are located in `openidm/ui/admin/default`. To customize the UI, copy this directory to `openidm/ui/admin/extension`:

```
cd /path/to/openidm/ui/admin
cp -r default/. extension
```

You can then set up custom files in the `extension` subdirectory.

The Admin UI templates in `openidm/ui/admin/default/templates` might help you get started.

+ *Default UI Subdirectories*

The Admin UI config files are located in separate subdirectories. The following list helps you locate the correct file(s) to customize:

`config/`

Top-level configuration directory of JavaScript files. Customizable subdirectories include `errorhandlers/` with HTTP error messages and `messages/` with info and error messages. For actual messages, see the `translation.json` file in the `locales/en/` subdirectory.

`css/` **and** `libs/`

If you use a different bootstrap theme, replace the files in these directories and subdirectories.

`fonts/`

The font files in this directory are based on the Font Awesome CSS toolkit.

`images/` **and** `img/`

IDM uses the image files in these directories. You can replace these images with your own.

`locales/`

The `translation.json` file (in the `en/` subdirectory by default) contains the UI labels and messages.

`org/`

Source files for the Admin UI.

`partials/`

Includes partial components of HTML pages in the Admin UI, for assignments, authentication, connectors, dashboards, email, basic forms, login buttons, and so on.

`templates/`

The HTML templates for various UI pages. Note that these files are used by the UI. Do not change the template files in `ui/admin/default/`.

+ *Customize the UI Theme*

You can configure a few features of the UI in the `ui-themeconfig.json` file in your project's `conf` directory. However, to change most theme-related features of the UI, you must copy target files to the appropriate `extension` subdirectory, and then modify them as discussed in "Customize the Admin UI".

By default the UI reads the stylesheets and images from the `openidm/ui/admin/default` directory. Do not modify the files in this directory. Your changes may be overwritten the next time you update or even patch your system.

To customize your UI, first set up matching subdirectories in (`openidm/ui/admin/extension`). For example, `openidm/ui/admin/extension/libs` and `openidm/ui/admin/extension/css`.

You might also need to update the `"stylesheets"` listing in the `ui-themeconfig.json` file for your project, in the *`project-dir`*`/conf` directory.

```
"stylesheets" : [
    "css/bootstrap-3.4.1-custom.css",
    "css/structure.css",
    "css/theme.css"
],
```

The default stylesheets have the following function:

- `bootstrap-3.4.1-custom.css`: Includes custom settings that you can get from various Bootstrap configuration sites, such as the Bootstrap Customize and Download site. This site lets you upload a `config.json` file that makes it easier to create a customized Bootstrap file. The ForgeRock version of this file is in `ui/admin/default/css/common/structure/`. You can use this file as a starting point for your customization.
- `structure.css`: For configuring structural elements of the UI.
- `theme.css`: Includes customizable options for UI themes such as colors, buttons, and navigation bars.

To set up custom versions of these files, copy them to the `extension/css` subdirectories.

+ *Change the Default Logo*

The default UI logo is in `openidm/ui/admin/default/images`. To change the logo, place your custom image in the `openidm/ui/admin/extension/images` directory. You should see the changes after refreshing your browser.

To specify a different file name, or to control the size, and other properties of the logo image file, adjust the `logo` property in the UI theme configuration file for your project `(conf/ui-themeconfig.json)`.

The following change to the UI theme configuration file points to an image file named `example-logo.png`, in the `openidm/ui/admin/extension/images` directory:

```
...
"loginLogo" : {
     "src" : "images/example-logo.png",
     "title" : "Example.com",
     "alt" : "Example.com",
     "height" : "104px",
     "width" : "210px"
},
...
```

Refresh your browser window for the new logo to appear.

+ *Create Project-Specific Themes*

You can create different UI themes for projects and then point a particular UI instance to a defined theme on startup. To create a complete custom theme, follow these steps:

1. Shut down the IDM instance, if it is running.
2. Copy the entire default Admin UI theme to an accessible location. For example:

```
cp -r /path/to/openidm/ui/admin/default /path/to/openidm/admin-project-theme
```

3. In the copied theme, modify the required elements, as described in the previous sections. Note that nothing is copied to the extension folder in this case—changes are made in the copied theme.

   In the `conf/ui.context-admin.json` file, modify the values for `defaultDir` and `extensionDir` to the directory with your `new-project-theme`:

```
{
    "enabled" : true,
    "urlContextRoot" : "/",
    "defaultDir" : "&{idm.install.dir}/ui/admin/default",
    "extensionDir" : "&{idm.install.dir}/ui/admin/extension",
    "responseHeaders" : {
        "X-Frame-Options" : "DENY"
    }
}
```

4. Restart the server.
5. Relaunch the UI in your browser. The UI is displayed with the new custom theme.

+ *Set Custom Response Headers*

You can specify custom response headers for your UI by using the `responseHeaders` property in UI context configuration files such as `conf/ui.context-admin.json`. For example, the `X-Frame-Options` header is a security measure used to prevent a web page from being embedded within the frame of another page. For more information about response headers, see the MDN page on HTTP Headers.

Because the `responseHeaders` property is specified in the configuration file for each UI context, you can set different custom headers for different UIs. For example, you might different security headers included for the Admin and End User UIs.

+ *Disable the Admin and End User UIs*

The UIs are packaged as separate bundles that can be disabled in the configuration before server startup.

To disable the registration of the UI servlets, edit the *project-dir*/conf/ui.context-*ui*.json files, setting the `enabled` property to false. For example, to disable the End User UI, set `"enabled" : false,` in *project-dir*/conf/ui.context-enduser.json

Reset User Passwords

When working with end users, administrators frequently have to reset their passwords. You can do so directly, through the Admin UI. Alternatively, you can configure an external system for that purpose, or set up password reset, as described in "*Password Reset*" in the *Self-Service Reference*.

+ *Change User Passwords Through the Admin UI*

1. Select Manage > User and select the user whose password you want to change.
2. Select the Password tab for that user and change the password.

+ *Use an External Password Reset System*

By default, the Password Reset mechanism is handled within IDM. You can reroute Password Reset in the event that a user has forgotten their password, by specifying an external URL to which Password Reset requests are sent. Note that this URL applies to the Password Reset link on the login page only, not to the security data change facility that is available after a user has logged in.

To set an external URL to handle Password Reset, set the `passwordResetLink` parameter in `conf/ui-configuration.json`. The following example sets the `passwordResetLink` to `https://accounts.example.com/account/reset-password`:

```
passwordResetLink: "https://accounts.example.com/reset-password"
```

The `passwordResetLink` parameter takes either an empty string as a value (which indicates that no external link is used) or a full URL to the external system that handles Password Reset requests.

Note

External Password Reset and security questions for internal Password Reset are mutually exclusive. Therefore, if you set a value for the `passwordResetLink` parameter, users will not be prompted with any security questions, regardless of the setting of the `securityQuestions` parameter.

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 a good introduction, see the OSGi site. 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.