Developer's Guide

/ Directory Services 6.5

Latest update: 6.5.6

Mark Craig
Ludovic Poitou

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

Abstract

Guide to developing client applications, server extensions, and applications that embed servers by using ForgeRock® Directory Services software.

Table of Contents

Preface

This guide shows you how to use ForgeRock® APIs to develop client applications, server extensions, and applications that embed servers.

In reading and following the instructions in this guide, you will learn how to:

- Access directory services using REST APIs over HTTP
- Access directory services using the LDAP command-line tools
- Use LDAP schema
- Work with standard LDAP groups and Directory Services-specific groups
- Work with LDAP collective attributes and Directory Services virtual attributes
- Work with LDAP referrals in search results
- Develop custom directory service Java plugins
- Embed the server in a Java application

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

Understanding LDAP

This chapter introduces directory concepts and directory server features. In this chapter you will learn:

- Why directory services exist and what they do well
- How data is arranged in directories that support Lightweight Directory Access Protocol (LDAP)
- How clients and servers communicate in LDAP
- What operations are standard according to LDAP and how standard extensions to the protocol work

A directory resembles a dictionary or a phone book. If you know a word, you can look it up its entry in the dictionary to learn its definition or its pronunciation. If you know a name, you can look it up its entry in the phone book to find the telephone number and street address associated with the name. If you are bored, curious, or have lots of time, you can also read through the dictionary, phone book, or directory, entry after entry.

Where a directory differs from a paper dictionary or phone book is in how entries are indexed. Dictionaries typically have one index—words in alphabetical order. Phone books, too—names in alphabetical order. Directories' entries on the other hand are often indexed for multiple attributes, names, user identifiers, email addresses, and telephone numbers. This means you can look up a directory entry by the name of the user the entry belongs to, but also by their user identifier, their email address, or their telephone number, for example.

How Directories and LDAP Evolved

Phone companies have been managing directories for many decades. The Internet itself has relied on distributed directory services like DNS since the mid 1980s.

It was not until the late 1980s, however, that experts from what is now the International Telecommunications Union published the X.500 set of international standards, including Directory Access Protocol. The X.500 standards specify Open Systems Interconnect (OSI) protocols and data definitions for general purpose directory services. The X.500 standards were designed to meet the needs of systems built according to the X.400 standards, covering electronic mail services.

Lightweight Directory Access Protocol (LDAP) has been around since the early 1990s. LDAP was originally developed as an alternative protocol that would allow directory access over Internet protocols rather than OSI protocols, and be lightweight enough for desktop implementations. By the mid-1990s, LDAP directory servers became generally available and widely used.

Until the late 1990s, LDAP directory servers were designed primarily with quick lookups and high availability for lookups in mind. LDAP directory servers replicate data, so when an update is made, that update is applied to other peer directory servers. Thus, if one directory server goes down, lookups can continue on other servers. Furthermore, if a directory service needs to support more lookups, the administrator can simply add another directory server to replicate with its peers.

As organizations rolled out larger and larger directories serving more and more applications, they discovered that they needed high availability not only for lookups, but also for updates. Around the year 2000, directories began to support multi-master replication; that is, replication with multiple read-write servers. Soon thereafter the organizations with the very largest directories started to need higher update performance as well as availability.

The DS code base began in the mid-2000s, when engineers solving the update performance issue decided the cost of adapting the existing C-based directory technology for high-performance updates would be higher than the cost of building a new, high-performance directory using Java technology.

About Data In LDAP Directories

LDAP directory data is organized into entries, similar to the entries for words in the dictionary, or for subscriber names in the phone book. A sample entry follows:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
uid: bjensen
cn: Babs Jensen
cn: Barbara Jensen
facsimileTelephoneNumber: +1 408 555 1992
gidNumber: 1000
givenName: Barbara
homeDirectory: /home/bjensen
l: San Francisco
mail: bjensen@example.com
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: posixAccount
objectClass: top
ou: People
ou: Product Development
roomNumber: 0209
sn: Jensen
telephoneNumber: +1 408 555 1862
uidNumber: 1076
```

Barbara Jensen's entry has a number of attributes, such as `uid: bjensen`, `telephoneNumber: +1 408 555 1862`, and `objectClass: posixAccount`. (The `objectClass` attribute type indicates which types of attributes are required and allowed for the entry. As the entries object classes can be updated online, and even the definitions of object classes and attributes are expressed as entries that can be updated online, directory data is extensible on the fly.) When you look up her entry in the directory, you specify one

or more attributes and values to match. The directory server then returns entries with attribute values that match what you specified.

The attributes you search for are indexed in the directory, so the directory server can retrieve them more quickly. (Attribute values do not have to be strings. Some attribute values, like certificates and photos, are binary.

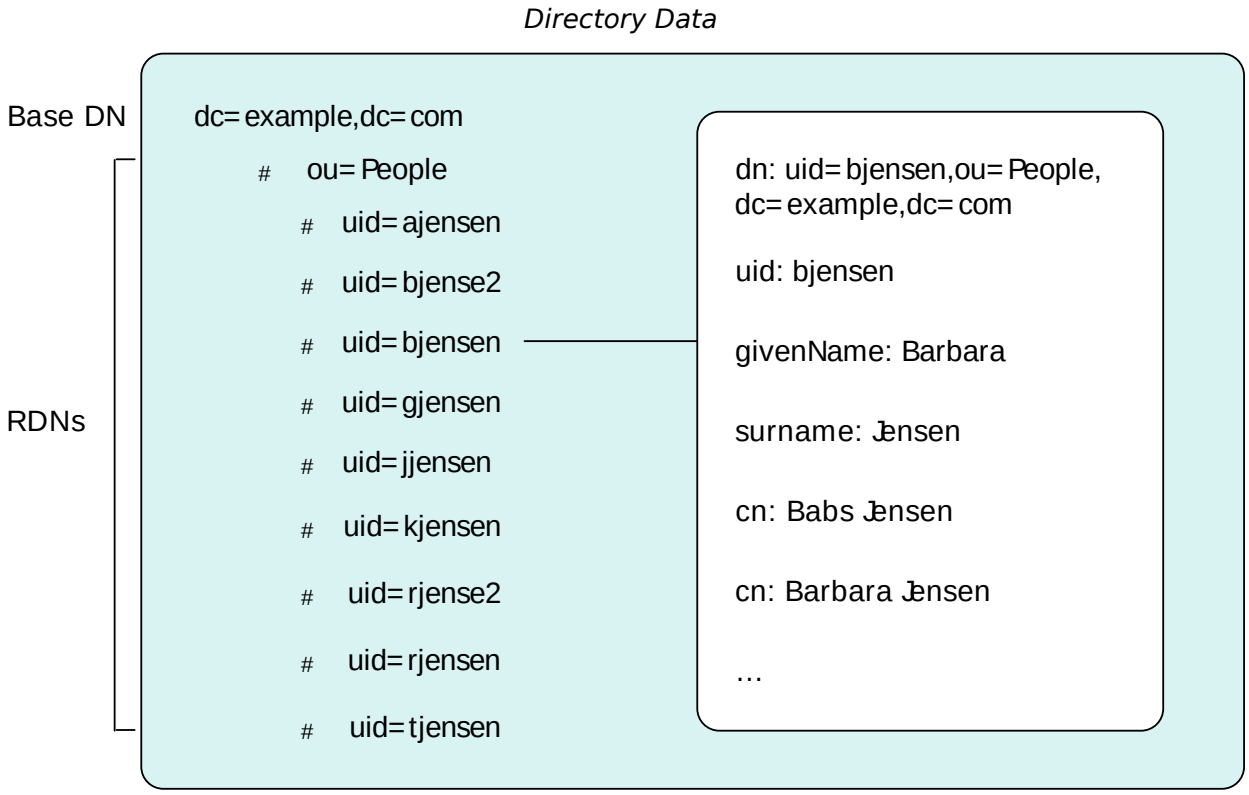
The entry also has a unique identifier, shown at the top of the entry, `dn: uid=bjensen,ou=People,dc=example,dc=com`. DN is an acronym for distinguished name. No two entries in the directory have the same distinguished name. Yet, DNs are typically composed of case-insensitive attributes.

Sometimes distinguished names include characters that you must escape. The following example shows an entry that includes escaped characters in the DN:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=escape)"
dn: cn=DN Escape Characters \" # \+ \, \; \< = \> \\,dc=example,dc=com
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: top
givenName: DN Escape Characters
uid: escape
cn: DN Escape Characters " # + , ; < = > \
sn: " # + , ; < = > \
mail: escape@example.com
```

LDAP entries are arranged hierarchically in the directory. The hierarchical organization resembles a file system on a PC or a web server, often imagined as an upside down tree structure, or a pyramid. The distinguished name consists of components separated by commas, `uid=bjensen,ou=People,dc=example,dc=com`. The names are little-endian. The components reflect the hierarchy of directory entries.

"Directory Data" shows the hierarchy.



Barbara Jensen's entry is located under an entry with DN `ou=People,dc=example,dc=com`, an organization unit and parent entry for the people at Example.com. The `ou=People` entry is located under the entry with DN `dc=example,dc=com`, the base entry for Example.com. DC is an acronym for domain component. The directory has other base entries, such as `cn=config`, under which the configuration is accessible through LDAP. A directory can serve multiple organizations, too. You might find `dc=example,dc=com`, `dc=mycompany,dc=com`, and `o=myOrganization` in the same LDAP directory. Therefore, when you look up entries, you specify the base DN to look under in the same way you need to know whether to look in the New York, Paris, or Tokyo phone book to find a telephone number. (The root entry for the directory, technically the entry with DN `""` (the empty string), is called the root DSE. It contains information about what the server supports, including the other base DNs it serves.

A directory server stores two kinds of attributes in a directory entry: *user attributes* and *operational attributes*. User attributes hold the information for users of the directory. All of the attributes shown in the entry at the outset of this section are user attributes. Operational attributes hold information used by the directory itself. Examples of operational attributes include `entryUUID`, `modifyTimestamp`, and `subschemaSubentry`. When an LDAP search operation finds an entry in the directory, the directory server returns all the visible user attributes unless the search request restricts the list of attributes by specifying those attributes explicitly. The directory server does not, however, return any operational

attributes unless the search request specifically asks for them. Generally speaking, applications should change only user attributes, and leave updates of operational attributes to the server, relying on public directory server interfaces to change server behavior. An exception is access control instruction (`aci`) attributes, which are operational attributes used to control access to directory data.

About LDAP Client and Server Communication

You may be used to web service client server communication, where each time the web client has something to request of the web server, a connection is set up and then torn down. LDAP has a different model. In LDAP the client application connects to the server and authenticates, then requests any number of operations, perhaps processing results in between requests, and finally disconnects when done.

The standard operations are as follows:

- Bind (authenticate). The first operation in an LDAP session usually involves the client binding to the LDAP server with the server authenticating the client. Authentication identifies the client's identity in LDAP terms, the identity which is later used by the server to authorize (or not) access to directory data that the client wants to lookup or change.

  If the client does not bind explicitly, the server treats the client as an anonymous client. An anonymous client is allowed to do anything that can be done anonymously. What can be done anonymously depends on access control and configuration settings. The client can also bind again on the same connection.

- Search (lookup). After binding, the client can request that the server return entries based on an LDAP filter, which is an expression that the server uses to find entries that match the request, and a base DN under which to search. For example, to look up all entries for people with email address `bjensen@example.com` in data for Example.com, you would specify a base DN such as `ou=People,dc=example,dc=com` and the filter `(mail=bjensen@example.com)`.

- Compare. After binding, the client can request that the server compare an attribute value the client specifies with the value stored on an entry in the directory.

- Modify. After binding, the client can request that the server change one or more attribute values stored on an entry. Often administrators do not allow clients to change directory data, so request that your administrator set appropriate access rights for your client application if you want to update data.

- Add. After binding, the client can request to add one or more new LDAP entries to the server.

- Delete. After binding, the client can request that the server delete one or more entries. To delete an entry with other entries underneath, first delete the children, then the parent.

- Modify DN. After binding, the client can request that the server change the distinguished name of the entry. In other words, this renames the entry or moves it to another location. For example, if Barbara changes her unique identifier from `bjensen` to something else, her DN would have to

change. For another example, if you decide to consolidate `ou=Customers` and `ou=Employees` under `ou=People` instead, all the entries underneath must change distinguished names.

Renaming entire branches of entries can be a major operation for the directory, so avoid moving entire branches if you can.

- Unbind. When done making requests, the client can request an unbind operation to end the LDAP session.
- Abandon. When a request seems to be taking too long to complete, or when a search request returns many more matches than desired, the client can send an abandon request to the server to drop the operation in progress.

About LDAP Controls and Extensions

LDAP has standardized two mechanisms for extending the operations directory servers can perform beyond the basic operations listed above. One mechanism involves using LDAP controls. The other mechanism involves using LDAP extended operations.

LDAP controls are information added to an LDAP message to further specify how an LDAP operation should be processed. For example, the server-side sort request control modifies a search to request that the directory server return entries to the client in sorted order. The subtree delete request control modifies a delete to request that the server also remove child entries of the entry targeted for deletion.

The directory server can also send response controls in some cases to indicate that the response contains special information. Examples include responses for entry change notification, password policy, and paged results.

LDAP extended operations are additional LDAP operations not included in the original standard list. For example, the cancel extended operation works like an abandon operation, but finishes with a response from the server after the cancel is complete. The StartTLS extended operation allows a client to connect to a server on an insecure port, but then start Transport Layer Security negotiations to protect communications.

Both LDAP controls and extended operations are demonstrated later in this guide. DS servers support many LDAP controls and a few LDAP extended operations, controls and extended operations matching those demonstrated in this guide.

Chapter 2

Best Practices For Application Developers

Follow the advice in this chapter to write effective, maintainable, high-performance directory client applications.

Authenticate Correctly

Unless your application performs only read operations, authenticate to the directory server. Some directory services require authentication to read directory data.

Once you authenticate (bind), directory servers make authorization decisions based on your identity. With servers that support proxied authorization, once authenticated, your application can request an operation on behalf of another identity, for example, the identity of the end user.

Your application therefore should have an account used to authenticate such as `cn=My App,ou=Apps,dc=example,dc=com`. The directory administrator can then authorize appropriate access for your application, and monitor your application's requests to help you troubleshoot problems if they arise.

Your application can use simple, password-based authentication. When using password-based authentication, also use secure connections to avoid sending the password as cleartext over the network. If you prefer to manage certificates rather than passwords, directory servers can managed certificate-based authentication as well.

Reuse Connections

LDAP is a stateful protocol. You authenticate (bind), you do stuff, you unbind. The server maintains a context that lets it make authorization decisions concerning your requests. You should therefore reuse connections when possible.

You can make multiple requests without having to set up a new connection and authenticate for every request. You can issue a request and get results asynchronously, while you issue another request. You can even share connections in a pool, avoiding the overhead of setting up and tearing down connections if you use them often.

Health Check Connections

In a network built for HTTP applications, your long-lived LDAP connections can get cut by network equipment configured to treat idle and even just old connections as stale resources to reclaim.

When you maintain a particularly long-lived connection such as a connection for a persistent search, periodically perform a health check to make sure nothing on the network quietly decided to drop your connection without notification. A health check might involve reading an attribute on a well-known entry in the directory.

Request Exactly What You Need All At Once

By the time your application makes it to production, you should know what attributes you want. Request them explicitly and request all the attributes in the same search.

Use Specific LDAP Filters

The difference between a general filter `(mail=*@example.com)` and a good, specific filter like `(mail=user@example.com)` can be huge numbers of entries and enormous amounts of processing time, both for the directory server that has to return search results, and also for your application that has to sort through the results. Many use cases can be handled with short, specific filters. As a rule, prefer equality filters over substring filters.

Some directory servers like DS servers reject unindexed searches by default, because unindexed searches are generally far more resource-intensive. If your application needs to use a filter that results in an unindexed search, then work with your directory administrator to find a solution, such as having the directory maintain the indexes required by your application.

Furthermore, always use `&` with `!` to restrict the potential result set before returning all entries that do not match part of the filter. For example, `(&(location=Oslo)(!(mail=birthday.girl@example.com)))`.

Make Modifications Specific

When you modify attributes with multiple values, for example, when you modify a list of group members, replace or delete specific values individually, rather than replacing the entire list of values. Making modifications specific helps directory servers replicate your changes more effectively.

Trust Result Codes

Trust the LDAP result code that your application gets from the directory server. For example, if you request a modify application and you get `ResultCode.SUCCESS`, then consider the operation a success rather than immediately issuing a search to get the modified entry.

The LDAP replication model is loosely convergent. In other words, the directory server can, and probably does, send you `ResultCode.SUCCESS` before replicating your change to every directory server

instance across the network. If you issue a read immediately after a write, and a load balancer sends your request to another directory server instance, you could get a result that differs from what you expect.

The loosely convergent model means that the entry could have changed since you read it. If needed, use LDAP assertions to set conditions for your LDAP operations.

Handle Input Securely

When taking input directly from a user or another program, handle the input securely by using appropriate methods to sanitize the data.

Failure to sanitize the input data can leave your application vulnerable to injection attacks.

Check Group Membership on the Account, Not the Group

If you need to determine which groups an account belongs to, request the DS virtual attribute, `isMemberOf`, when you read the account entry. Other directory servers use other names for this attribute that identifies the groups to an account belongs to.

Ask the Directory Server What It Supports

Directory servers expose their capabilities, suffixes they support, and other information as attribute values on the root DSE.

This allows your application to discover a variety of information at run time, rather than storing configuration separately. Thus putting effort into querying the directory about its configuration and the features it supports can make your application easier to deploy and to maintain.

For example, rather than hard-coding `dc=example,dc=com` as a suffix DN in your configuration, you can search the root DSE on DS servers for `namingContexts`, and then search under the naming context DNs to locate the desired entries to initialize your configuration.

Directory servers also expose their schema over LDAP. The root DSE attribute `subschemaSubentry` shows the DN of the entry holding LDAP schema definitions. Note that LDAP object class and attribute type names are case-insensitive, so `isMemberOf` and `ismemberof` refer to the same attribute, for example.

Store Large Attribute Values By Reference

When you use large attribute values such as photos or audio messages, consider storing the objects themselves elsewhere and keeping only a reference to external content on directory entries. In order to serve results quickly with high availability, directory servers both cache content and also replicate it everywhere.

Textual entries with a bunch of attributes and perhaps a certificate are often no larger than a few KB. Your directory administrator might therefore be disappointed to learn that your popular application stores users' photo and .mp3 collections as attributes of their accounts.

Take Care With Persistent Search and Server-Side Sorting

A persistent search lets your application receive updates from the server as they happen by keeping the connection open and forcing the server to check whether to return additional results any time it performs a modification in the scope of your search. Directory administrators therefore might hesitate to grant persistent search access to your application. Directory servers like DS servers can let you discover updates with less overhead by searching the change log periodically. If you do have to use a persistent search instead, try to narrow the scope of your search.

Directory servers also support a resource-intensive operation called server-side sorting. When your application requests a server-side sort, the directory server retrieves all the entries matching your search, and then returns the whole set of entries in sorted order. For result sets of any size server-side sorting therefore ties up server resources that could be used elsewhere. Alternatives include both sorting the results after your application receives them, and working with the directory administrator to enable appropriate browsing (virtual list view) indexes on the directory server for applications that must regularly page through long lists of search results.

Reuse Schemas Where Possible

Directory servers like DS servers come with schema definitions for a wide range of standard object classes and attribute types. This is because directories are designed to be shared by many applications. Directories use unique, typically IANA-registered object identifiers (OIDs) to avoid object class and attribute type name clashes. The overall goal is Internet-wide interoperability.

You therefore should reuse schema definitions that already exist whenever you reasonably can. Reuse them as is. Do not try to redefine existing schema definitions.

If you must add schema definitions for your application, extend existing object classes with AUXILIARY classes of your own. Take care to name your definitions such that they do not clash with other names.

When you have defined schema required for your application, work with the directory administrator to add your definitions to the directory service. Directory servers like DS servers let directory administrators update schema definitions over LDAP, so there is not generally a need to interrupt the service to add your application. Directory administrators can, however, have other reasons why they hesitate to add your schema definitions. Coming to the discussion prepared with good schema definitions, explanations of why they should be added, and evident regard for interoperability makes it easier for the directory administrator to grant your request.

Handle Referrals

When a directory server returns a search result, the result is not necessarily an entry. If the result is a referral, then your application should follow up with an additional search based on the URIs provided in the result.

Troubleshooting: Check Result Codes

LDAP result codes are standard and clearly defined, and listed in "*LDAP Result Codes*". When your application receives a result, it must the result code value to determine what action to take. When the result is not what you expect, read or at least log the additional message information.

Troubleshooting: Check Server Log Files

If you can read the directory server access log, then you can check what the server did with your application's request. For example, the following access log excerpt shows a successful search by `cn=My App,ou=Apps,dc=example,dc=com`. The lines are wrapped for readability, whereas in the log each record starts with the timestamp:


```
{"eventName":"DJ-LDAP","client":{"ip":"127.0.0.1","port":59891},"server":
{"ip":"127.0.0.1","port":1389},"request":
{"protocol":"LDAP","operation":"CONNECT","connId":0},"transactionId":"0","response":
{"status":"SUCCESSFUL","statusCode":"0","elapsedTime":0,"elapsedTimeUnits":"MILLISECONDS"},"timestamp":"2016-10-20T
a640-805db1c35af0-3"}
{"eventName":"DJ-LDAP","client":{"ip":"127.0.0.1","port":59891},"server":
{"ip":"127.0.0.1","port":1389},"request":
{"protocol":"LDAP","operation":"EXTENDED","connId":0,"msgId":1,"name":"StartTLS","oid":"1.3.6.1.4.1.1466.20037"},"t
{"status":"SUCCESSFUL","statusCode":"0","elapsedTime":3,"elapsedTimeUnits":"MILLISECONDS"},"timestamp":"2016-10-20T
a640-805db1c35af0-5"}
{"eventName":"DJ-LDAP","client":{"ip":"127.0.0.1","port":59891},"server":
{"ip":"127.0.0.1","port":1389},"request":
{"protocol":"LDAP","operation":"BIND","connId":0,"msgId":2,"version":"3","authType":"Simple","dn":"cn=My
 App,ou=Apps,dc=example,dc=com"},"transactionId":"0","response":
{"status":"SUCCESSFUL","statusCode":"0","elapsedTime":6,"elapsedTimeUnits":"MILLISECONDS"},"userId":"cn=My
 App,ou=Apps,dc=example,dc=com","timestamp":"2016-10-20T15:48:38.462Z","_id":"11d5fdaf-79ac-4677-
a640-805db1c35af0-7"}
{"eventName":"DJ-LDAP","client":{"ip":"127.0.0.1","port":59891},"server":
{"ip":"127.0.0.1","port":1389},"request":
{"protocol":"LDAP","operation":"SEARCH","connId":0,"msgId":3,"dn":"dc=example,dc=com","scope":"sub","filter":"(uid=
["isMemberOf"]},"transactionId":"0","response":
{"status":"SUCCESSFUL","statusCode":"0","elapsedTime":6,"elapsedTimeUnits":"MILLISECONDS","nentries":1},"timestamp"
a640-805db1c35af0-9"}
{"eventName":"DJ-LDAP","client":{"ip":"127.0.0.1","port":59891},"server":
{"ip":"127.0.0.1","port":1389},"request":
{"protocol":"LDAP","operation":"UNBIND","connId":0,"msgId":4},"transactionId":"0","timestamp":"2016-10-20T15:48:38.
a640-805db1c35af0-11"}
{"eventName":"DJ-LDAP","client":{"ip":"127.0.0.1","port":59891},"server":
{"ip":"127.0.0.1","port":1389},"request":
{"protocol":"LDAP","operation":"DISCONNECT","connId":0},"transactionId":"0","response":
{"status":"SUCCESSFUL","statusCode":"0","elapsedTime":0,"elapsedTimeUnits":"MILLISECONDS","reason":"Client
 Unbind"},"timestamp":"2016-10-20T15:48:38.481Z","_id":"11d5fdaf-79ac-4677-a640-805db1c35af0-13"}
```

Notice that each operation type is shown in upper case. The messages track the client information, and the connection ID (`connId`) and message ID (`msgID`) numbers for filtering messages. The `elapsedTime` indicates how many milliseconds the DS server worked on the request. Result code 0 corresponds to a successful result, as described in RFC 4511.

Troubleshooting: Inspect Network Traffic

If result codes and server logs are not enough, many network tools can interpret HTTP and LDAP packets. Install the necessary keys to decrypt encrypted packet content.

Chapter 3

Performing RESTful Operations

DS software lets you access directory data as JSON resources over HTTP. DS software maps JSON resources onto LDAP entries. As a result, REST clients perform many of the same operations as LDAP clients with directory data.

This chapter demonstrates RESTful client operations by using a directory server for evaluation as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*.

In this chapter, you will learn how to use the DS REST API that provides access to directory data over HTTP. In particular, you will learn how to:

- Create a resource that does not yet exist
- Read a single resource
- Update an existing resource
- Delete an existing resource
- Patch part of an existing resource
- Perform a predefined action
- Query a set of resources
- Work with other MIME types for resources like photos

For examples configuration REST APIs, see "*Configuring REST APIs*" in the *Administration Guide*.

Before trying the examples, enable HTTP access to the DS server as described in "To Set Up REST Access to User Data" in the *Administration Guide*. The examples in this chapter use HTTP, but the procedure also shows how to set up HTTPS access to the server.

The DS REST API is built on a common ForgeRock HTTP-based REST API for interacting with JSON Resources. All APIs built on this common layer let you perform the following operations.

About ForgeRock Common REST

ForgeRock® Common REST is a common REST API framework. It works across the ForgeRock platform to provide common ways to access web resources and collections of resources. Adapt the examples in this section to your resources and deployment.

Note

This section describes the full Common REST framework. Some platform component products do not implement all Common REST behaviors exactly as described in this section. For details, refer to the product-specific examples and reference information in other sections of this documentation set.

Common REST Resources

Servers generally return JSON-format resources, though resource formats can depend on the implementation.

Resources in collections can be found by their unique identifiers (IDs). IDs are exposed in the resource URIs. For example, if a server has a user collection under `/users`, then you can access a user at `/users/user-id`. The ID is also the value of the `_id` field of the resource.

Resources are versioned using revision numbers. A revision is specified in the resource's `_rev` field. Revisions make it possible to figure out whether to apply changes without resource locking and without distributed transactions.

Common REST Verbs

The Common REST APIs use the following verbs, sometimes referred to collectively as CRUDPAQ. For details and HTTP-based examples of each, follow the links to the sections for each verb.

Create

Add a new resource.

This verb maps to HTTP PUT or HTTP POST.

For details, see "Create".

Read

Retrieve a single resource.

This verb maps to HTTP GET.

For details, see "Read".

Update

Replace an existing resource.

This verb maps to HTTP PUT.

For details, see "Update".

Delete

Remove an existing resource.

This verb maps to HTTP DELETE.

For details, see "Delete".

Patch

Modify part of an existing resource.

This verb maps to HTTP PATCH.

For details, see "Patch".

Action

Perform a predefined action.

This verb maps to HTTP POST.

For details, see "Action".

Query

Search a collection of resources.

This verb maps to HTTP GET.

For details, see "Query".

Common REST Parameters

Common REST reserved query string parameter names start with an underscore, `_`.

Reserved query string parameters include, but are not limited to, the following names:

```
_action
_api
_crestapi
_fields
_mimeType
_pageSize
_pagedResultsCookie
_pagedResultsOffset
_prettyPrint
_queryExpression
```

`_queryFilter`
`_queryId`
`_sortKeys`
`_totalPagedResultsPolicy`

Note

Some parameter values are not safe for URLs, so URL-encode parameter values as necessary.

Continue reading for details about how to use each parameter.

Common REST Extension Points

The *action* verb is the main vehicle for extensions. For example, to create a new user with HTTP POST rather than HTTP PUT, you might use `/users?_action=create`. A server can define additional actions. For example, `/tasks/1?_action=cancel`.

A server can define *stored queries* to call by ID. For example, `/groups?_queryId=hasDeletedMembers`. Stored queries can call for additional parameters. The parameters are also passed in the query string. Which parameters are valid depends on the stored query.

Common REST API Documentation

Common REST APIs often depend at least in part on runtime configuration. Many Common REST endpoints therefore serve *API descriptors* at runtime. An API descriptor documents the actual API as it is configured.

Use the following query string parameters to retrieve API descriptors:

`_api`

Serves an API descriptor that complies with the OpenAPI specification.

This API descriptor represents the API accessible over HTTP. It is suitable for use with popular tools such as Swagger UI.

`_crestapi`

Serves a native Common REST API descriptor.

This API descriptor provides a compact representation that is not dependent on the transport protocol. It requires a client that understands Common REST, as it omits many Common REST defaults.

Note

Consider limiting access to API descriptors in production environments in order to avoid unnecessary traffic.

To provide documentation in production environments, see "To Publish OpenAPI Documentation" instead.

To Publish OpenAPI Documentation

In production systems, developers expect stable, well-documented APIs. Rather than retrieving API descriptors at runtime through Common REST, prepare final versions, and publish them alongside the software in production.

Use the OpenAPI-compliant descriptors to provide API reference documentation for your developers as described in the following steps:

1. Configure the software to produce production-ready APIs.

   In other words, the software should be configured as in production so that the APIs are identical to what developers see in production.

2. Retrieve the OpenAPI-compliant descriptor.

   The following command saves the descriptor to a file, `myapi.json`:

   ```
   $ curl -o myapi.json endpoint?_api
   ```

3. (Optional) If necessary, edit the descriptor.

   For example, you might want to add security definitions to describe how the API is protected.

   If you make any changes, then also consider using a source control system to manage your versions of the API descriptor.

4. Publish the descriptor using a tool such as Swagger UI.

   You can customize Swagger UI for your organization as described in the documentation for the tool.

Create

There are two ways to create a resource, either with an HTTP POST or with an HTTP PUT.

To create a resource using POST, perform an HTTP POST with the query string parameter `_action=create` and the JSON resource as a payload. Accept a JSON response. The server creates the identifier if not specified:

```
POST /users?_action=create HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
{ JSON resource }
```

To create a resource using PUT, perform an HTTP PUT including the case-sensitive identifier for the resource in the URL path, and the JSON resource as a payload. Use the `If-None-Match: *` header. Accept a JSON response:

```
PUT /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-None-Match: *
{ JSON resource }
```

The `_id` and content of the resource depend on the server implementation. The server is not required to use the `_id` that the client provides. The server response to the create request indicates the resource location as the value of the `Location` header.

If you include the `If-None-Match` header, its value must be `*`. In this case, the request creates the object if it does not exist, and fails if the object does exist. If you include the `If-None-Match` header with any value other than `*`, the server returns an HTTP 400 Bad Request error. For example, creating an object with `If-None-Match: revision` returns a bad request error. If you do not include `If-None-Match: *`, the request creates the object if it does not exist, and *updates* the object if it does exist.

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Read

To retrieve a single resource, perform an HTTP GET on the resource by its case-sensitive identifier (`_id`) and accept a JSON response:

```
GET /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
```

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

`_mimeType=mime-type`

Some resources have fields whose values are multi-media resources such as a profile photo for example.

If the feature is enabled for the endpoint, you can read a single field that is a multi-media resource by specifying the *field* and *mime-type*.

In this case, the content type of the field value returned matches the *mime-type* that you specify, and the body of the response is the multi-media resource.

The `Accept` header is not used in this case. For example, `Accept: image/png` does not work. Use the `_mimeType` query string parameter instead.

Update

To update a resource, perform an HTTP PUT including the case-sensitive identifier (`_id`) as the final element of the path to the resource, and the JSON resource as the payload. Use the `If-Match: _rev` header to check that you are actually updating the version you modified. Use `If-Match: *` if the version does not matter. Accept a JSON response:

```
PUT /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-Match: _rev
{ JSON resource }
```

When updating a resource, include all the attributes to be retained. Omitting an attribute in the resource amounts to deleting the attribute unless it is not under the control of your application. Attributes not under the control of your application include private and read-only attributes. In addition, virtual attributes and relationship references might not be under the control of your application.

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Delete

To delete a single resource, perform an HTTP DELETE by its case-sensitive identifier (`_id`) and accept a JSON response:

```
DELETE /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
```

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Patch

To patch a resource, send an HTTP PATCH request with the following parameters:

- `operation`
- `field`
- `value`
- `from` (optional with copy and move operations)

You can include these parameters in the payload for a PATCH request, or in a JSON PATCH file. If successful, you'll see a JSON response similar to:

```
PATCH /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-Match: _rev
{ JSON array of patch operations }
```

PATCH operations apply to three types of targets:

- **single-valued**, such as an object, string, boolean, or number.
- **list semantics array**, where the elements are ordered, and duplicates are allowed.
- **set semantics array**, where the elements are not ordered, and duplicates are not allowed.

ForgeRock PATCH supports several different `operations`. The following sections show each of these operations, along with options for the `field` and `value`:

Patch Operation: Add

The `add` operation ensures that the target field contains the value provided, creating parent fields as necessary.

If the target field is single-valued, then the value you include in the PATCH replaces the value of the target. Examples of a single-valued field include: object, string, boolean, or number.

An `add` operation has different results on two standard types of arrays:

- **List semantic arrays**: you can run any of these `add` operations on that type of array:
  - If you `add` an array of values, the PATCH operation appends it to the existing list of values.
  - If you `add` a single value, specify an ordinal element in the target array, or use the `{-}` special index to add that value to the end of the list.
- **Set semantic arrays**: The value included in the patch is merged with the existing set of values. Any duplicates within the array are removed.

As an example, start with the following list semantic array resource:

```
{
    "fruits" : [ "orange", "apple" ]
}
```

The following add operation includes the pineapple to the end of the list of fruits, as indicated by the `-` at the end of the `fruits` array.

```
{
    "operation" : "add",
    "field" : "/fruits/-",
    "value" : "pineapple"
}
```

The following is the resulting resource:

```
{
    "fruits" : [ "orange", "apple", "pineapple" ]
}
```

Note that you can add only one array element one at a time, as per the corresponding JSON Patch specification. If you add an array of elements, for example:

```
{
    "operation" : "add",
    "field" : "/fruits/-",
    "value" : ["pineapple", "mango"]
}
```

The resulting resource would have the following invalid JSON structure:

```
{
    "fruits" : [ "orange", "apple", ["pineapple", "mango"]]
}
```

Patch Operation: Copy

The copy operation takes one or more existing values from the source field. It then adds those same values on the target field. Once the values are known, it is equivalent to performing an `add` operation on the target field.

The following `copy` operation takes the value from a field named `mail`, and then runs a `replace` operation on the target field, `another_mail`.

```
[
  {
     "operation":"copy",
     "from":"mail",
     "field":"another_mail"
  }
]
```

If the source field value and the target field value are configured as arrays, the result depends on whether the array has list semantics or set semantics, as described in "Patch Operation: Add".

Patch Operation: Increment

The `increment` operation changes the value or values of the target field by the amount you specify. The value that you include must be one number, and may be positive or negative. The value of the target field must accept numbers. The following `increment` operation adds `1000` to the target value of `/user/payment`.

```
[
  {
    "operation" : "increment",
    "field" : "/user/payment",
    "value" : "1000"
  }
]
```

Since the `value` of the `increment` is a single number, arrays do not apply.

Patch Operation: Move

The move operation removes existing values on the source field. It then adds those same values on the target field. It is equivalent to performing a `remove` operation on the source, followed by an `add` operation with the same values, on the target.

The following `move` operation is equivalent to a `remove` operation on the source field, `surname`, followed by a `replace` operation on the target field value, `lastName`. If the target field does not exist, it is created.

```
[
  {
     "operation":"move",
     "from":"surname",
     "field":"lastName"
  }
]
```

To apply a `move` operation on an array, you need a compatible single-value, list semantic array, or set semantic array on both the source and the target. For details, see the criteria described in "Patch Operation: Add".

Patch Operation: Remove

The `remove` operation ensures that the target field no longer contains the value provided. If the remove operation does not include a value, the operation removes the field. The following `remove` deletes the value of the `phoneNumber`, along with the field.

```
[
  {
    "operation" : "remove",
    "field" : "phoneNumber"
  }
]
```

If the object has more than one `phoneNumber`, those values are stored as an array.

A `remove` operation has different results on two standard types of arrays:

- **List semantic arrays**: A `remove` operation deletes the specified element in the array. For example, the following operation removes the first phone number, based on its array index (zero-based):

  ```
  [
     {
        "operation" : "remove",
        "field" : "/phoneNumber/0"
     }
  ]
  ```

- **Set semantic arrays**: The list of values included in a patch are removed from the existing array.

Patch Operation: Replace

The `replace` operation removes any existing value(s) of the targeted field, and replaces them with the provided value(s). It is essentially equivalent to a `remove` followed by a `add` operation. If the arrays are used, the criteria is based on "Patch Operation: Add". However, indexed updates are not allowed, even when the target is an array.

The following `replace` operation removes the existing `telephoneNumber` value for the user, and then adds the new value of `+1 408 555 9999`.

```
[
  {
    "operation" : "replace",
    "field" : "/telephoneNumber",
    "value" : "+1 408 555 9999"
  }
]
```

A PATCH replace operation on a list semantic array works in the same fashion as a PATCH remove operation. The following example demonstrates how the effect of both operations. Start with the following resource:

```
{
    "fruits" : [ "apple", "orange", "kiwi", "lime" ],
}
```

Apply the following operations on that resource:

```
[
  {
    "operation" : "remove",
    "field" : "/fruits/0",
    "value" : ""
  },
  {
    "operation" : "replace",
    "field" : "/fruits/1",
    "value" : "pineapple"
  }
]
```

The PATCH operations are applied sequentially. The `remove` operation removes the first member of that resource, based on its array index, (`fruits/0`), with the following result:

```
[
  {
    "fruits" : [ "orange", "kiwi", "lime" ],
  }
]
```

The second PATCH operation, a `replace`, is applied on the second member (`fruits/1`) of the intermediate resource, with the following result:

```
[
  {
    "fruits" : [ "orange", "pineapple", "lime" ],
  }
]
```

Patch Operation: Transform

The `transform` operation changes the value of a field based on a script or some other data transformation command. The following `transform` operation takes the value from the field named `/objects`, and applies the `something.js` script as shown:

```
[
  {
    "operation" : "transform",
    "field" : "/objects",
    "value" : {
      "script" : {
        "type" : "text/javascript",
        "file" : "something.js"
      }
    }
  }
]
```

Patch Operation Limitations

Some HTTP client libraries do not support the HTTP PATCH operation. Make sure that the library you use supports HTTP PATCH before using this REST operation.

For example, the Java Development Kit HTTP client does not support PATCH as a valid HTTP method. Instead, the method `HttpURLConnection.setRequestMethod("PATCH")` throws `ProtocolException`.

Parameters

You can use the following parameters. Other parameters might depend on the specific action implementation:

`_prettyPrint=true`

Format the body of the response.

`_fields=`*`field`*`[,`*`field`*`...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Action

Actions are a means of extending Common REST APIs and are defined by the resource provider, so the actions you can use depend on the implementation.

The standard action indicated by `_action=create` is described in "Create".

Parameters

You can use the following parameters. Other parameters might depend on the specific action implementation:

`_prettyPrint=true`

Format the body of the response.

`_fields=`*`field`*`[,`*`field`*`...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

Query

To query a resource collection (or resource container if you prefer to think of it that way), perform an HTTP GET and accept a JSON response, including at least a `_queryExpression`, `_queryFilter`, or `_queryId` parameter. These parameters cannot be used together:

```
GET /users?_queryFilter=true HTTP/1.1
Host: example.com
Accept: application/json
```

The server returns the result as a JSON object including a "results" array and other fields related to the query string parameters that you specify.

Parameters

You can use the following parameters:

`_queryFilter=`*`filter-expression`*

Query filters request that the server return entries that match the filter expression. You must URL-escape the filter expression.

The string representation is summarized as follows. Continue reading for additional explanation:

```
Expr           = OrExpr
OrExpr         = AndExpr ( 'or' AndExpr ) *
AndExpr        = NotExpr ( 'and' NotExpr ) *
NotExpr        = '!' PrimaryExpr | PrimaryExpr
PrimaryExpr    = '(' Expr ')' | ComparisonExpr | PresenceExpr | LiteralExpr
ComparisonExpr = Pointer OpName JsonValue
PresenceExpr   = Pointer 'pr'
LiteralExpr    = 'true' | 'false'
Pointer        = JSON pointer
OpName         = 'eq' |  # equal to
                 'co' |  # contains
                 'sw' |  # starts with
                 'lt' |  # less than
                 'le' |  # less than or equal to
                 'gt' |  # greater than
                 'ge' |  # greater than or equal to
                 STRING  # extended operator
JsonValue      = NUMBER | BOOLEAN | '"' UTF8STRING '"'
STRING         = ASCII string not containing white-space
UTF8STRING     = UTF-8 string possibly containing white-space
```

JsonValue components of filter expressions follow RFC 7159: *The JavaScript Object Notation (JSON) Data Interchange Format*. In particular, as described in section 7 of the RFC, the escape

character in strings is the backslash character. For example, to match the identifier `test\`, use `_id eq 'test\\'`. In the JSON resource, the `\` is escaped the same way: `"_id":"test\\"`.

When using a query filter in a URL, be aware that the filter expression is part of a query string parameter. A query string parameter must be URL encoded as described in RFC 3986: *Uniform Resource Identifier (URI): Generic Syntax* For example, white space, double quotes (`"`), parentheses, and exclamation characters need URL encoding in HTTP query strings. The following rules apply to URL query components:

```
query       = *( pchar / "/" / "?" )
pchar       = unreserved / pct-encoded / sub-delims / ":" / "@"
unreserved  = ALPHA / DIGIT / "-" / "." / "_" / "~"
pct-encoded = "%" HEXDIG HEXDIG
sub-delims  = "!" / "$" / "&" / "'" / "(" / ")"
                  / "*" / "+" / "," / ";" / "="
```

`ALPHA`, `DIGIT`, and `HEXDIG` are core rules of RFC 5234: *Augmented BNF for Syntax Specifications*:

```
ALPHA       =  %x41-5A / %x61-7A   ; A-Z / a-z
DIGIT       =  %x30-39             ; 0-9
HEXDIG      =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
```

As a result, a backslash escape character in a *JsonValue* component is percent-encoded in the URL query string parameter as `%5C`. To encode the query filter expression `_id eq 'test\\'`, use `_id +eq+'test%5C%5C'`, for example.

A simple filter expression can represent a comparison, presence, or a literal value.

For comparison expressions use *json-pointer comparator json-value*, where the *comparator* is one of the following:

`eq` (equals)
`co` (contains)
`sw` (starts with)
`lt` (less than)
`le` (less than or equal to)
`gt` (greater than)
`ge` (greater than or equal to)

For presence, use *json-pointer pr* to match resources where:

- The JSON pointer is present.
- The value it points to is not `null`.

Literal values include true (match anything) and false (match nothing).

Complex expressions employ `and`, `or`, and `!` (not), with parentheses, `(expression)`, to group expressions.

`_queryId=identifier`

Specify a query by its identifier.

Specific queries can take their own query string parameter arguments, which depend on the implementation.

`_pagedResultsCookie=string`

The string is an opaque cookie used by the server to keep track of the position in the search results. The server returns the cookie in the JSON response as the value of `pagedResultsCookie`.

In the request `_pageSize` must also be set and non-zero. You receive the cookie value from the provider on the first request, and then supply the cookie value in subsequent requests until the server returns a `null` cookie, meaning that the final page of results has been returned.

The `_pagedResultsCookie` parameter is supported when used with the `_queryFilter` parameter. The `_pagedResultsCookie` parameter is not guaranteed to work when used with the `_queryExpression` and `_queryId` parameters.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and not to be used together.

`_pagedResultsOffset=integer`

When `_pageSize` is non-zero, use this as an index in the result set indicating the first page to return.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and not to be used together.

`_pageSize=integer`

Return query results in pages of this size. After the initial request, use `_pagedResultsCookie` or `_pageResultsOffset` to page through the results.

`_totalPagedResultsPolicy=string`

When a `_pageSize` is specified, and non-zero, the server calculates the "totalPagedResults", in accordance with the `totalPagedResultsPolicy`, and provides the value as part of the response. The "totalPagedResults" is either an estimate of the total number of paged results (`_totalPagedResultsPolicy=ESTIMATE`), or the exact total result count (`_totalPagedResultsPolicy=EXACT`). If no count policy is specified in the query, or if `_totalPagedResultsPolicy=NONE`, result counting is disabled, and the server returns value of -1 for "totalPagedResults".

`_sortKeys=[+-]field[,[+-]field...]`

Sort the resources returned based on the specified field(s), either in `+` (ascending, default) order, or in `-` (descending) order.

Because ascending order is the default, including the `+` character in the query is unnecessary. If you do include the `+`, it must be URL-encoded as `%2B`, for example:

```
http://localhost:8080/api/users?_prettyPrint=true&_queryFilter=true&_sortKeys=%2Bname/givenName
```

The `_sortKeys` parameter is not supported for predefined queries (`_queryId`).

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in each element of the "results" array in the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

HTTP Status Codes

When working with a Common REST API over HTTP, client applications should expect at least the following HTTP status codes. Not all servers necessarily return all status codes identified here:

200 OK

The request was successful and a resource returned, depending on the request.

201 Created

The request succeeded and the resource was created.

204 No Content

The action request succeeded, and there was no content to return.

304 Not Modified

The read request included an `If-None-Match` header, and the value of the header matched the revision value of the resource.

400 Bad Request

The request was malformed.

401 Unauthorized

The request requires user authentication.

403 Forbidden

Access was forbidden during an operation on a resource.

404 Not Found

The specified resource could not be found, perhaps because it does not exist.

405 Method Not Allowed

The HTTP method is not allowed for the requested resource.

406 Not Acceptable

The request contains parameters that are not acceptable, such as a resource or protocol version that is not available.

409 Conflict

The request would have resulted in a conflict with the current state of the resource.

410 Gone

The requested resource is no longer available, and will not become available again. This can happen when resources expire for example.

412 Precondition Failed

The resource's current version does not match the version provided.

415 Unsupported Media Type

The request is in a format not supported by the requested resource for the requested method.

428 Precondition Required

The resource requires a version, but no version was supplied in the request.

500 Internal Server Error

The server encountered an unexpected condition that prevented it from fulfilling the request.

501 Not Implemented

The resource does not support the functionality required to fulfill the request.

503 Service Unavailable

The requested resource was temporarily unavailable. The service may have been disabled, for example.

Selecting an API Version

DS REST APIs can be versioned. If there is more than one version of the API, then you must select the version by setting a version header that specifies which version of the resource is requested:

```
Accept-API-Version: resource=version
```

Here, *version* is the value of the `version` field in the mapping configuration file for the API. For details, see "Mapping Configuration File" in the *Reference*.

If you do not set a version header, then the latest version is returned.

The default example configuration includes only one API, whose version is `1.0`. In this case, the header can be omitted. If used in the examples below, the appropriate header would be `Accept-API-Version: resource=1.0`.

Authenticating Over REST

When you first try to read a resource that can be read as an LDAP entry with an anonymous search, you learn that you must authenticate as shown in the following example:

```
$ curl http://opendj.example.com:8080/api/users/bjensen
{"code":401,"reason":"Unauthorized","message":"Invalid Credentials"}
```

HTTP status code 401 indicates that the request requires user authentication.

To prevent DS servers from requiring authentication, set the Rest2ldap endpoint `authorization-mechanism` to map anonymous HTTP requests to LDAP requests performed by an authorized user, as in the following example that uses Kirsten Vaughan's identity:

```
$ dsconfig \
 set-http-authorization-mechanism-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --mechanism-name "HTTP Anonymous" \
 --set enabled:true \
 --set user-dn:uid=kvaughan,ou=people,dc=example,dc=com \
 --trustAll \
 --no-prompt
$ dsconfig \
 set-http-endpoint-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --endpoint-name "/api" \
 --set authorization-mechanism:"HTTP Anonymous" \
 --trustAll \
 --no-prompt
```

By default, both the Rest2ldap endpoint and also the REST to LDAP gateway allow HTTP Basic authentication and HTTP header-based authentication in the style of OpenIDM software. The

authentication mechanisms translate HTTP authentication to LDAP authentication to the directory server.

When you set up a directory server for evaluation or with generated sample data, the relative distinguished name (DN) attribute for sample user entries is the user ID (`uid`) attribute. For example, the DN and user ID for Babs Jensen are:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
uid: bjensen
```

Given this pattern in the user entries, the default REST to LDAP configuration translates the HTTP user name to the LDAP user ID. User entries are found directly under `ou=People,dc=example,dc=com`. (In general, REST to LDAP mappings require that LDAP entries mapped to JSON resources be immediate subordinates of the mapping's baseDN.) In other words, Babs Jensen authenticates as `bjensen` (password: `hifalutin`) over HTTP. The corresponding LDAP bind DN is `uid=bjensen,ou=People,dc=example,dc=com`.

HTTP Basic authentication works as shown in the following example:

```
$ curl \
 --user bjensen:hifalutin \
 http://opendj.example.com:8080/api/users/bjensen?_fields=userName
{"_id":"bjensen","_rev":"<revision>","userName":"bjensen@example.com"}
```

The alternative HTTP Basic *username*:*password*@ form in the URL works as shown in the following example:

```
$ curl \
 http://bjensen:hifalutin@opendj.example.com:8080/api/users/bjensen?_fields=userName
{"_id":"bjensen","_rev":"<revision>","userName":"bjensen@example.com"}
```

HTTP header based authentication works as shown in the following example:

```
$ curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: hifalutin" \
 http://opendj.example.com:8080/api/users/bjensen?_fields=userName
{"_id":"bjensen","_rev":"<revision>","userName":"bjensen@example.com"}
```

If the directory data is laid out differently or if the user names are email addresses rather than user IDs, for example, then you must update the configuration in order for authentication to work.

The REST to LDAP gateway can also translate HTTP user name and password authentication to LDAP PLAIN SASL authentication. Likewise, the gateway falls back to proxied authorization as necessary,

using connections to LDAP servers on which the directory superuser has authenticated. See "*REST to LDAP Configuration*" in the *Reference* for details on all configuration choices.

Creating Resources

There are two alternative ways to create resources:

- To create a resource using an ID that you specify, perform an HTTP PUT request with headers `Content-Type: application/json` and `If-None-Match: *`, and the JSON content of your resource.

  The following example shows you how to create a new user entry with ID `newuser`:

```
$ curl \
 --request PUT \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --header "If-None-Match: *" \
 --data '{
  "_id": "newuser",
  "_schema":"frapi:opendj:rest2ldap:user:1.0",
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1212",
    "emailAddress": "newuser@example.com"
  },
  "name": {
    "familyName": "New",
    "givenName": "User"
  },
  "displayName": ["New User"],
  "manager": {
    "_id": "kvaughan",
    "displayName": "Kirsten Vaughan"
  }
 }' \
 http://opendj.example.com:8080/api/users/newuser?_prettyPrint=true
{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName" : "User",
    "familyName" : "New"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "manager" : {
    "_id" : "kvaughan",
```

```
    "displayName" : "Kirsten Vaughan"
  }
}
```

- To create a resource and let the server choose the ID, perform an HTTP POST with `_action=create` as described in "Using Actions".

Reading a Resource

To read a resource, perform an HTTP GET as shown in the following example:

```
$ curl \
 --request GET \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/api/users/newuser?_prettyPrint=true
{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName" : "User",
    "familyName" : "New"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "manager" : {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  }
}
```

Updating Resources

To update a resource, replace it by performing an HTTP PUT including the case-sensitive identifier (`_id`) as the final element of the path to the resource, and a JSON payload that contains *all* writable fields of the resource that you want to retain. Use an `If-Match` header to ensure the resource already exists. For read-only fields, either include unmodified versions, or omit them from your updated version.

To update a resource regardless of the revision, use an `If-Match: *` header. The following example writes a new entry with an additional display name for Sam Carter:

```
$ curl \
 --request PUT \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --header "If-Match: *" \
 --data '{
   "contactInformation": {
     "telephoneNumber": "+1 408 555 4798",
     "emailAddress": "scarter@example.com"
   },
   "name": {
     "familyName": "Carter",
     "givenName": "Sam"
   },
   "userName": "scarter@example.com",
   "displayName": ["Sam Carter", "Samantha Carter"],
   "groups": [
     {
       "_id": "Accounting Managers"
     }
   ],
   "manager": {
     "_id": "trigden",
     "displayName": "Torrey Rigden"
   },
  "uidNumber": 1002,
  "gidNumber": 1000,
  "homeDirectory": "/home/scarter"
 }' \
 http://opendj.example.com:8080/api/users/scarter?_prettyPrint=true
{
  "_id" : "scarter",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "userName" : "scarter@example.com",
  "displayName" : [ "Sam Carter", "Samantha Carter" ],
  "name" : {
    "givenName" : "Sam",
    "familyName" : "Carter"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 4798",
    "emailAddress" : "scarter@example.com"
  },
  "uidNumber" : 1002,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/scarter",
  "groups" : [ {
    "_id" : "Accounting Managers"
  } ],
  "manager" : {
    "_id" : "trigden",
    "displayName" : "Torrey Rigden"
  }
```

```
}
```

To update a resource only if the resource matches a particular version, use an `If-Match: revision` header as shown in the following example:

```
$ export REVISION=$(cut -d \" -f 8 <(curl --silent \
  --user kvaughan:bribery \
  http://opendj.example.com:8080/api/users/scarter?_fields=_rev))
$ curl \
 --request PUT \
 --user kvaughan:bribery \
 --header "If-Match: $REVISION" \
 --header "Content-Type: application/json" \
 --data '{
   "_id" : "scarter",
   "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
   "contactInformation": {
     "telephoneNumber": "+1 408 555 4798",
     "emailAddress": "scarter@example.com"
   },
   "name": {
     "familyName": "Carter",
     "givenName": "Sam"
   },
   "userName": "scarter@example.com",
   "displayName": ["Sam Carter", "Samantha Carter"],
  "uidNumber": 1002,
  "gidNumber": 1000,
  "homeDirectory": "/home/scarter"
 }' \
 http://opendj.example.com:8080/api/users/scarter?_prettyPrint=true
{
  "_id" : "scarter",
  "_rev" : "<new-revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "userName" : "scarter@example.com",
  "displayName" : [ "Sam Carter", "Samantha Carter" ],
  "name" : {
    "givenName" : "Sam",
    "familyName" : "Carter"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 4798",
    "emailAddress" : "scarter@example.com"
  },
  "uidNumber" : 1002,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/scarter",
  "groups" : [ {
    "_id" : "Accounting Managers"
  } ]
}
```

Deleting Resources

To delete a resource, perform an HTTP DELETE on the resource URL. The operation returns the resource you deleted as shown in the following example:

```
$ curl \
 --request DELETE \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/api/users/newuser?_prettyPrint=true
{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName" : "User",
    "familyName" : "New"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "manager" : {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  }
}
```

To delete a resource only if the resource matches a particular version, use an `If-Match: revision` header as shown in the following example:

```
$ export REVISION=$(cut -d \" -f 8 <(curl --silent \
  --user kvaughan:bribery \
  http://opendj.example.com:8080/api/users/newuser?_fields=_rev))
$ curl \
 --request DELETE \
 --user kvaughan:bribery \
 --header "If-Match: $REVISION" \
 http://opendj.example.com:8080/api/users/newuser?_prettyPrint=true
{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName" : "User",
```

```
    "familyName" : "New"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "manager" : {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  }
}
```

To delete a resource and all of its children, you must change the configuration, get the REST to LDAP gateway or Rest2ldap endpoint to reload its configuration, and perform the operation as a user who has the access rights required. The following steps show one way to do this with the Rest2ldap endpoint.

In this example, the LDAP view of the user to delete shows two child entries as seen in the following example:

```
$ ldapsearch --port 1389 --baseDN uid=nbohr,ou=people,dc=example,dc=com "(&)" 1.1
dn: uid=nbohr,ou=People,dc=example,dc=com

dn: cn=quantum dot,uid=nbohr,ou=People,dc=example,dc=com

dn: cn=qubit generator,uid=nbohr,ou=People,dc=example,dc=com
```

1. If you are using the gateway, this requires the default setting of true for `useSubtreeDelete` in `WEB-INF/classes/rest2ldap/rest2ldap.json`.

   Note

   Only users who have access to request a tree delete can delete resources with children.

2. Force the Rest2ldap to reread its configuration as shown in the following **dsconfig** commands:

```
$ dsconfig \
 set-http-endpoint-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:false \
 --trustAll \
 --no-prompt
$ dsconfig \
 set-http-endpoint-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

3. Allow the REST user to use the subtree delete control:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --add global-aci:"(targetcontrol=\"1.2.840.113556.1.4.805\")\
 (version 3.0; acl \"Allow Subtree Delete\"; allow(read) \
 userdn=\"ldap:///uid=kvaughan,ou=People,dc=example,dc=com\";)" \
 --trustAll \
 --no-prompt
```

4. Request the delete as a user who has rights to perform a subtree delete on the resource as shown in the following example:

```
$ curl \
 --request DELETE \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/api/users/nbohr?_prettyPrint=true
{
  "_id" : "nbohr",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : { },
  "userName" : "nbohr@example.com",
  "displayName" : [ "Niels Bohr" ],
  "name" : {
    "givenName" : "Niels",
    "familyName" : "Bohr"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "nbohr@example.com"
  },
  "uidNumber" : 1111,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/nbohr"
}
```

Patching Resources

DS software lets you patch JSON resources, updating part of the resource rather than replacing it. For example, you could change Babs Jensen's email address by issuing an HTTP PATCH request as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "replace",
    "field": "/contactInformation/emailAddress",
    "value": "babs@example.com"
  }
 ]' \
 http://opendj.example.com:8080/api/users/bjensen?_prettyPrint=true
{
  "_id" : "bjensen",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "userName" : "babs@example.com",
  "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
  "name" : {
```

```
    "givenName" : "Barbara",
    "familyName" : "Jensen"
  },
  "description" : "Original description",
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1862",
    "emailAddress" : "babs@example.com"
  },
  "uidNumber" : 1076,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/bjensen",
  "manager" : {
    "_id" : "trigden",
    "displayName" : "Torrey Rigden"
  },
  "groups" : [ {
    "_id" : "Carpoolers"
  } ]
}
```

Notice in the example that the data sent specifies the type of patch operation, the field to change, and a value that depends on the field you change and on the operation. A single-valued field takes an object, boolean, string, or number depending on its type, whereas a multi-valued field takes an array of values. Getting the type wrong results in an error. Also notice that the patch data is itself an array. This makes it possible to patch more than one part of the resource by using a set of patch operations in the same request.

DS software supports four types of patch operations:

`add`

The add operation ensures that the target field contains the value provided, creating parent fields as necessary.

If the target field is single-valued and a value already exists, then that value is replaced with the value you provide. *Note that you do not get an error when adding a value to a single-valued field that already has a value.* A single-valued field is one whose value is not an array (an object, string, boolean, or number).

If the target field is multi-valued, then the array of values you provide is merged with the set of values already in the resource. New values are added, and duplicate values are ignored. A multi-valued field takes an array value.

`remove`

The remove operation ensures that the target field does not contain the value provided. If you do not provide a value, the entire field is removed if it already exists.

If the target field is single-valued and a value is provided, then the provided value must match the existing value to remove, otherwise the field is left unchanged.

If the target field is multi-valued, then values in the array you provide are removed from the existing set of values.

`replace`

The replace operation removes existing values on the target field, and replaces them with the values you provide. It is equivalent to performing a remove on the field, then an add with the values you provide.

`increment`

The increment operation increments or decrements the value or values in the target field by the amount you specify, which is positive to increment and negative to decrement. The target field must take a number or a set of numbers. The value you provide must be a single number.

One key nuance in how a patch works with DS software concerns multi-valued fields. Although JSON resources represent multi-valued fields as *arrays*, DS software treats those values as *sets*. In other words, values in the field are unique, and the ordering of an array of values is not meaningful in the context of patch operations. If you reference array values by index, DS software returns an error. DS software does, however, allow use of a hyphen to add an element to a set. Include the hyphen as the last element of the `field` JSON pointer path as in `"/members/-"` in this example patch: `[{ "operation" : "add", "field" : "/members/-", "value" : { "_id" : "bjensen" } }]`.

Perform patch operations as if arrays values were sets. The following example includes Barbara Jensen in a group by adding her to the set of members:

```
$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --header "Content-Type: application/json" \
 --data '[{
     "operation": "add",
     "field": "/members",
     "value": [{"_id": "bjensen"}]
   }]' \
 http://opendj.example.com:8080/api/groups/Directory%20Administrators?_prettyPrint=true
{
  "_id" : "Directory Administrators",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:group:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "displayName" : "Directory Administrators",
  "members" : [ {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  }, {
    "_id" : "rdaugherty",
    "displayName" : "Robert Daugherty"
  }, {
    "_id" : "hmiller",
    "displayName" : "Harry Miller"
  }, {
    "_id" : "bjensen",
    "displayName" : [ "Barbara Jensen", "Babs Jensen" ]
  } ]
}
```

The following example removes Barbara Jensen from the group:

```
$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --header "Content-Type: application/json" \
 --data '[{
    "operation": "remove",
    "field": "/members",
    "value": [{"_id": "bjensen"}]
  }]' \
 http://opendj.example.com:8080/api/groups/Directory%20Administrators?_prettyPrint=true
{
  "_id" : "Directory Administrators",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:group:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "displayName" : "Directory Administrators",
  "members" : [ {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  }, {
    "_id" : "rdaugherty",
    "displayName" : "Robert Daugherty"
  }, {
    "_id" : "hmiller",
    "displayName" : "Harry Miller"
  } ]
}
```

To change the value of more than one attribute in a patch operation, include multiple operations in the body of the JSON patch, as shown in the following example:

```
$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "replace",
    "field": "/contactInformation/telephoneNumber",
    "value": "+1 408 555 9999"
  },
  {
    "operation": "add",
    "field": "/contactInformation/emailAddress",
    "value": "barbara.jensen@example.com"
  }
 ]' \
 http://opendj.example.com:8080/api/users/bjensen?_prettyPrint=true
{
  "_id" : "bjensen",
```

```
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : {
    "lastModified" : "<datestamp>"
  },
  "userName" : "barbara.jensen@example.com",
  "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
  "name" : {
    "givenName" : "Barbara",
    "familyName" : "Jensen"
  },
  "description" : "Original description",
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 9999",
    "emailAddress" : "barbara.jensen@example.com"
  },
  "uidNumber" : 1076,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/bjensen",
  "manager" : {
    "_id" : "trigden",
    "displayName" : "Torrey Rigden"
  },
  "groups" : [ {
    "_id" : "Carpoolers"
  } ]
}
```

Notice that for a multi-valued attribute, the `value` field takes an array, whereas the `value` field takes a single value for a single-valued field. Also notice that for single-valued fields, an `add` operation has the same effect as a `replace` operation.

You can use resource revision numbers in `If-Match: revision` headers to patch the resource only if the resource matches a particular version, as shown in the following example:

```
$ export REVISION=$(cut -d \" -f 8 <(curl --silent \
  --user kvaughan:bribery \
  http://opendj.example.com:8080/api/users/bjensen?_fields=_rev))
$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --header "If-Match: $REVISION" \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "add",
    "field": "/contactInformation/emailAddress",
    "value": "babs@example.com"
  }
 ]' \
 http://opendj.example.com:8080/api/users/bjensen?_prettyPrint=true
{
  "_id" : "bjensen",
  "_rev" : "<new-revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : {
```

```
    "lastModified" : "<datestamp>"
  },
  "userName" : "babs@example.com",
  "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
  "name" : {
    "givenName" : "Barbara",
    "familyName" : "Jensen"
  },
  "description" : "Original description",
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 9999",
    "emailAddress" : "babs@example.com"
  },
  "uidNumber" : 1076,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/bjensen",
  "groups" : [ {
    "_id" : "Carpoolers"
  } ],
  "manager" : {
    "_id" : "trigden",
    "displayName" : "Torrey Rigden"
  }
}
```

The resource revision changes when the patch is successful.

Using Actions

DS REST to LDAP implements the actions described in this section.

Using the Create Resource Action

DS software implements an action that lets the server set the resource ID on creation. To use this action, perform an HTTP POST with header `Content-Type: application/json`, and the JSON content of the resource.

The `_action=create` in the query string is optional.

The following example creates a new user entry:

```
$ curl \
 --request POST \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --data '{
  "_id": "newuser",
  "_schema": "frapi:opendj:rest2ldap:user:1.0",
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1212",
    "emailAddress": "newuser@example.com"
```

```
  },
  "name": {
    "familyName": "New",
    "givenName": "User"
  },
  "displayName": ["New User"],
  "manager": {"_id": "kvaughan", "displayName": "Kirsten Vaughan"}
 }' \
 http://opendj.example.com:8080/api/users?_prettyPrint=true
{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<datestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName" : "User",
    "familyName" : "New"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "manager" : {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  }
}
```

Using the Modify Password and Reset Password Actions

DS software implements actions for resetting and changing passwords.

These actions require HTTPS to avoid sending passwords over insecure connections. Before trying the examples that follow, enable HTTPS on the HTTP connection handler as described in "RESTful Client Access Over HTTP" in the *Administration Guide*. Notice that the following examples use the exported server certificate, `server-cert.pem`, generated in that procedure. If the connection handler uses a certificate signed by a well-known CA, then you can omit the `--cacert` option.

Changing Passwords

The `modifyPassword` action lets a user modify their password given the old password and a new password.

To use this action, perform an HTTP POST over HTTPS with header `Content-Type: application/json`, `_action=modifyPassword` in the query string, and the old and new passwords in JSON format as the POST data.

The JSON must include the following fields:

`oldPassword`

The value of this field is the current password as a UTF-8 string.

`newPassword`

The value of this field is the new password as a UTF-8 string.

The following example demonstrates a user changing their own password. On success, the HTTP status code is 200 OK, and the response body is an empty JSON resource:

```
$ curl \
 --request POST \
 --cacert server-cert.pem \
 --user bjensen:hifalutin \
 --header "Content-Type: application/json" \
 --data '{"oldPassword": "hifalutin", "newPassword": "password"}' \
 https://opendj.example.com:8443/api/users/bjensen?_action=modifyPassword
{}
```

Resetting Passwords

The `resetPassword` action lets a user or password administrator reset a password to a generated password value.

To use this action, perform an HTTP POST over HTTPS with header `Content-Type: application/json`, `_action=resetPassword` in the query string, and an empty JSON document (`{}`) as the POST data.

The following example demonstrates an administrator changing a user's password. Before trying this example, make sure the password administrator user has been given the `password-reset` privilege as shown in "To Add Privileges for an Individual Administrator" in the *Administration Guide*. Otherwise, the password administrator has insufficient access. On success, the HTTP status code is 200 OK, and the response body is a JSON resource with a `generatedPassword` containing the new password:

```
$ curl \
 --request POST \
 --cacert server-cert.pem \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --data '{}' \
 https://opendj.example.com:8443/api/users/bjensen?_action=resetPassword
{"generatedPassword":"<new-password>"}
```

The password administrator communicates the new, generated password to the user.

This feature could be used in combination with a password policy that forces the user to change their password after a reset. For an example of such a policy, see "Require Password Change on Add or Reset" in the *Administration Guide*.

Querying Resource Collections

To query resource collections, perform an HTTP GET with a `_queryFilter=expression` parameter in the query string. For details about the query filter *expression*, see "Query".

The `_queryId`, and `_totalPagedResultsPolicy` parameters described in "Query" are not used in DS software at present.

"LDAP Search and REST Query Filters" shows some LDAP search filters with corresponding examples of query filter expressions.

LDAP Search and REST Query Filters

LDAP Filter	REST Filter
(&)	_queryFilter=true
(uid=*)	_queryFilter=_id+pr
(uid=bjensen)	_queryFilter=_id+eq+'bjensen'
(uid=*jensen*)	_queryFilter=_id+co+'jensen'
(uid=jensen*)	_queryFilter=_id+sw+'jensen'
(&(uid=*jensen*)(cn=babs*))	_queryFilter=(_id+co+'jensen'+and+displayName +sw+'babs')
(|(uid=*jensen*)(cn=sam*))	_queryFilter=(_id+co+'jensen'+or+displayName+sw +'sam')
(!(uid=*jensen*))	_queryFilter=!(_id+co+'jensen')
(uid<=jensen)	_queryFilter=_id+le+'jensen'
(uid>=jensen)	_queryFilter=_id+ge+'jensen'

For query operations, the filter *expression* is constructed from the following building blocks. Make sure you URL-encode the filter expressions, which are shown here without URL-encoding to make them easier to read.

In filter expressions, the simplest *json-pointer* is a field of the JSON resource, such as `userName` or `id`. A *json-pointer* can also point to nested elements as described in the JSON Pointer Internet-Draft:

Comparison expressions

Build filters using the following comparison expressions:

`json-pointer` `eq` `json-value`

Matches when the pointer equals the value, as in the following example:

```
$ curl \
```

```
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+eq
+'babs@example.com'&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "bjensen",
    "_rev" : "<revision>",
    "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
    "_meta" : {
      "lastModified" : "<datestamp>"
    },
    "userName" : "babs@example.com",
    "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
    "name" : {
      "givenName" : "Barbara",
      "familyName" : "Jensen"
    },
    "description" : "Original description",
    "manager" : {
      "_id" : "trigden",
      "displayName" : "Torrey Rigden"
    },
    "contactInformation" : {
      "telephoneNumber" : "+1 408 555 9999",
      "emailAddress" : "babs@example.com"
    },
    "uidNumber" : 1076,
    "gidNumber" : 1000,
    "homeDirectory" : "/home/bjensen",
    "groups" : [ {
      "_id" : "Carpoolers"
    } ]
  } ],
  "resultCount" : 1,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

json-pointer co json-value

Matches when the pointer contains the value, as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+co
+'jensen'&_fields=userName&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "ajensen",
    "_rev" : "<revision>",
    "userName" : "ajensen@example.com"
  }, {
    "_id" : "gjensen",
    "_rev" : "<revision>",
```

```
    "userName" : "gjensen@example.com"
  }, {
    "_id" : "jjensen",
    "_rev" : "<revision>",
    "userName" : "jjensen@example.com"
  }, {
    "_id" : "kjensen",
    "_rev" : "<revision>",
    "userName" : "kjensen@example.com"
  }, {
    "_id" : "rjensen",
    "_rev" : "<revision>",
    "userName" : "rjensen@example.com"
  }, {
    "_id" : "tjensen",
    "_rev" : "<revision>",
    "userName" : "tjensen@example.com"
  } ],
  "resultCount" : 6,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

`json-pointer sw json-value`

Matches when the pointer starts with the value, as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+sw
+'ab'&_fields=userName&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "abarnes",
    "_rev" : "<revision>",
    "userName" : "abarnes@example.com"
  }, {
    "_id" : "abergin",
    "_rev" : "<revision>",
    "userName" : "abergin@example.com"
  } ],
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

`json-pointer lt json-value`

Matches when the pointer is less than the value, as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+lt
+'ac'&_fields=userName&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "abarnes",
    "_rev" : "<revision>",
    "userName" : "abarnes@example.com"
  }, {
    "_id" : "abergin",
    "_rev" : "<revision>",
    "userName" : "abergin@example.com"
  } ],
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

`json-pointer` `le` `json-value`

Matches when the pointer is less than or equal to the value, as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+le
+'ad'&_fields=userName&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "abarnes",
    "_rev" : "<revision>",
    "userName" : "abarnes@example.com"
  }, {
    "_id" : "abergin",
    "_rev" : "<revision>",
    "userName" : "abergin@example.com"
  }, {
    "_id" : "achassin",
    "_rev" : "<revision>",
    "userName" : "achassin@example.com"
  } ],
  "resultCount" : 3,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

`json-pointer` `gt` `json-value`

Matches when the pointer is greater than the value, as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+gt
+'tt'&_fields=userName&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "ttully",
    "_rev" : "<revision>",
    "userName" : "ttully@example.com"
  }, {
    "_id" : "tward",
    "_rev" : "<revision>",
    "userName" : "tward@example.com"
  }, {
    "_id" : "wlutz",
    "_rev" : "<revision>",
    "userName" : "wlutz@example.com"
  } ],
  "resultCount" : 3,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

`json-pointer ge json-value`

Matches when the pointer is greater than or equal to the value, as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+ge
+'tw'&_fields=userName&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "tward",
    "_rev" : "<revision>",
    "userName" : "tward@example.com"
  }, {
    "_id" : "wlutz",
    "_rev" : "<revision>",
    "userName" : "wlutz@example.com"
  } ],
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

Presence expression

`json-pointer pr` matches any resource on which the *json-pointer* is present, as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/groups?_queryFilter=displayName
+pr&_fields=displayName&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "Accounting Managers",
    "_rev" : "<revision>",
    "displayName" : "Accounting Managers"
  }, {
    "_id" : "Directory Administrators",
    "_rev" : "<revision>",
    "displayName" : "Directory Administrators"
  }, {
    "_id" : "HR Managers",
    "_rev" : "<revision>",
    "displayName" : "HR Managers"
  }, {
    "_id" : "PD Managers",
    "_rev" : "<revision>",
    "displayName" : "PD Managers"
  }, {
    "_id" : "QA Managers",
    "_rev" : "<revision>",
    "displayName" : "QA Managers"
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

Literal expressions

`true` matches any resource in the collection.

`false` matches no resource in the collection.

In other words, you can list all resources in a collection as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/groups?_queryFilter=true&_fields=_id&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "Accounting Managers",
    "_rev" : "<revision>"
  }, {
    "_id" : "Directory Administrators",
    "_rev" : "<revision>"
  }, {
    "_id" : "HR Managers",
    "_rev" : "<revision>"
  }, {
    "_id" : "PD Managers",
    "_rev" : "<revision>"
  }, {
    "_id" : "QA Managers",
    "_rev" : "<revision>"
  }, {
    "_rev" : "<revision>"
  } ],
  "resultCount" : 6,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

Complex expressions

Combine expressions using boolean operators `and`, `or`, and `!` (not), and by using parentheses `(expression)` with group expressions. The following example queries resources with last name Jensen and manager name starting with `Bar`:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=\
(userName+co+'jensen'+and+manager/displayName+sw+'Sam')&_fields=displayName&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "jjensen",
    "_rev" : "<revision>",
    "displayName" : [ "Jody Jensen" ]
  }, {
    "_id" : "tjensen",
    "_rev" : "<revision>",
    "displayName" : [ "Ted Jensen" ]
  } ],
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

Notice that the filters use the JSON pointers `name/familyName` and `manager/displayName` to identify the fields nested inside the `name` and `manager` objects.

Paged Results

You can page through search results using the following query string parameters that are further described in "Query":

- `_pagedResultsCookie=`*`string`*
- `_pagedResultsOffset=`*`integer`*
- `_pageSize=`*`integer`*

The following example demonstrates how paged results are used:

```
# Request five results per page, and retrieve the first page.
$ curl \
 --user bjensen:hifalutin \
 "http://opendj.example.com:8080/api/users?_queryFilter=true&_fields=userName&_pageSize=5&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "abarnes",
    "_rev" : "<revision>",
    "userName" : "abarnes@example.com"
  }, {
    "_id" : "abergin",
    "_rev" : "<revision>",
    "userName" : "abergin@example.com"
  }, {
    "_id" : "achassin",
    "_rev" : "<revision>",
```

```
    "userName" : "achassin@example.com"
  }, {
    "_id" : "ahall",
    "_rev" : "<revision>",
    "userName" : "ahall@example.com"
  }, {
    "_id" : "ahel",
    "_rev" : "<revision>",
    "userName" : "ahel@example.com"
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : "<cookie>",
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
$ export COOKIE=$(cut -d \" -f 4 <(grep pagedResultsCookie \
 <(curl --silent --user bjensen:hifalutin \
 "http://opendj.example.com:8080/api/users?_queryFilter=true&_fields=userName&_pageSize=5&_prettyPrint=true")))

# Provide the cookie to request the next five results.
$ curl \
 --user bjensen:hifalutin \
 "http://opendj.example.com:8080/api/users?_queryFilter=true&_fields=userName&_pageSize=5\
&_pagedResultsCookie=$COOKIE&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "ahunter",
    "_rev" : "<revision>",
    "userName" : "ahunter@example.com"
  }, {
    "_id" : "ajensen",
    "_rev" : "<revision>",
    "userName" : "ajensen@example.com"
  }, {
    "_id" : "aknutson",
    "_rev" : "<revision>",
    "userName" : "aknutson@example.com"
  }, {
    "_id" : "alangdon",
    "_rev" : "<revision>",
    "userName" : "alangdon@example.com"
  }, {
    "_id" : "alutz",
    "_rev" : "<revision>",
    "userName" : "alutz@example.com"
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : "<new-cookie>",
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}

# Request the tenth page of five results.
$ curl \
 --user bjensen:hifalutin \
 "http://opendj.example.com:8080/api/users?_queryFilter=true&_fields=userName\
&_pageSize=5&_pagedResultsOffset=10&_prettyPrint=true"
```

```
{
  "result" : [ {
    "_id" : "ewalker",
    "_rev" : "<revision>",
    "userName" : "ewalker@example.com"
  }, {
    "_id" : "eward",
    "_rev" : "<revision>",
    "userName" : "eward@example.com"
  }, {
    "_id" : "falbers",
    "_rev" : "<revision>",
    "userName" : "falbers@example.com"
  }, {
    "_id" : "gfarmer",
    "_rev" : "<revision>",
    "userName" : "gfarmer@example.com"
  }, {
    "_id" : "gjensen",
    "_rev" : "<revision>",
    "userName" : "gjensen@example.com"
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : "<cookie>",
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

Notice the following features of the responses:

- `"remainingPagedResults" : -1` means that the number of remaining results is unknown.
- `"totalPagedResults" : -1` means that the total number of paged results is unknown.
- `"totalPagedResultsPolicy" : "NONE"` means that result counting is disabled.

Server-Side Sort

You can use the `_sortKeys` parameter, described in "Query", to request that the server sort the results it returns.

The following example sorts results by given name:

```
$ curl \
 --user bjensen:hifalutin \
 "http://opendj.example.com:8080/api/users?_queryFilter=name/familyName+eq+'jensen'\
&_sortKeys=name/givenName&_fields=name&_prettyPrint=true"
{
  "result" : [ {
    "_id" : "ajensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Allison",
```

```
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "bjensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Barbara",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "bjense2",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Bjorn",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "gjensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Gern",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "jjensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Jody",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "kjensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Kurt",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "rjense2",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Randy",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "rjensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Richard",
      "familyName" : "Jensen"
    }
  }, {
    "_id" : "tjensen",
    "_rev" : "<revision>",
    "name" : {
      "givenName" : "Ted",
      "familyName" : "Jensen"
    }
  } ],
```

```
  "resultCount" : 9,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

To sort in reverse order, use `_sortKeys=-field`.

To specify multiple sort keys, use a comma-separated list of fields.

The sort key fields that you specify must exist in the result entries.

As mentioned in "Take Care With Persistent Search and Server-Side Sorting", servers must store and then sort the result set for your search. If you expect a large result set for your search, use paged results, described in "Paged Results", to limit the impact on the server and get your results more quickly.

Working With Alternative Content Types

DS software generally maps JSON resources to LDAP entries. Some resources such as profile photos, however, are best expressed with other MIME types. ForgeRock® Common REST lets your applications make HTTP multipart requests, so you can work with other MIME types differently from regular JSON resources. This is done using the `_mimeType` parameter described in "Read".

This section includes the following procedures:

- "To Map an Alternative Content Type"
- "To Update a Non-JSON Resource"
- "To Read a Non-JSON Resource"

Note

The default configuration described in "To Set Up REST Access to User Data" in the *Administration Guide* does not include any mappings that require alternative content types. You must therefore add a mapping to use an alternative content type and disable and then enable the Rest2ldap endpoint for the change to take effect.

To Map an Alternative Content Type

To add a mapping to the configuration, follow these steps:

1. Edit the attributes section for a resource in the configuration file `/path/to/opendj/config/rest2ldap/endpoints/api/example-v1.json` to include a user property that maps to a MIME type.

   The following line adds a simple mapping from the `photo` property to the `jpegPhoto` LDAP attribute:

```
"photo" : { "type": "simple", "ldapAttribute" : "jpegPhoto" },
```

2. Force the Rest2ldap endpoint to reread the updated configuration file.

   You can force the Rest2ldap endpoint to reread its configuration by disabling it and then enabling it:

```
$ dsconfig \
 set-http-endpoint-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:false \
 --trustAll \
 --no-prompt
$ dsconfig \
 set-http-endpoint-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --endpoint-name "/api" \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

To Update a Non-JSON Resource

With a mapping configured as described in "To Map an Alternative Content Type", REST client applications can update MIME resources with form-based content as described in the following steps:

1. Ensure that the application has a resource to upload.

   For example, copy a JPEG photo `picture.jpg` to the current directory.

2. Upload the non-JSON resource with its metadata as a multipart form.

   The following example patches Babs Jensen's resource to add a profile photo:

```
$ curl \
 --request PATCH \
 --form 'json=[{"operation": "add", "field": "/photo",
        "value": {"$ref":"cid:picture#content"}}];type=application/json' \
 --form 'picture=@picture.jpg;type=image/jpeg' \
 "http://kvaugan:bribery@opendj.example.com:8080/api/users/bjensen?_prettyPrint=true"
{
  "_id" : "bjensen",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta" : {
```

```
    "lastModified" : "<datestamp>"
  },
  "userName" : "babs@example.com",
  "displayName" : [ "Barbara Jensen", "Babs Jensen" ],
  "name" : {
    "givenName" : "Barbara",
    "familyName" : "Jensen"
  },
  "description" : "Original description",
  "photo" : "<base64-photo>",
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 9999",
    "emailAddress" : "babs@example.com"
  },
  "uidNumber" : 1076,
  "gidNumber" : 1000,
  "homeDirectory" : "/home/bjensen",
  "manager" : {
    "_id" : "trigden",
    "displayName" : "Torrey Rigden"
  },
  "groups" : [ {
    "_id" : "Carpoolers"
  } ]
}
```

Notice the **curl** command form data. When you specify the reference to the content ID, the reference takes the form:

```
{"$ref":"cid:identifier#(content|filename|mimetype)"}
```

If you want other attributes to hold the filename (`picture.jpg`) and MIME type (`image/jpeg`) of the file you upload, you can reference those as well. In the example above, `{"$ref":"cid:picture#filename"}` is `picture.jpg` and `{"$ref":"cid:picture#mimetype"}` is `image/jpeg`.

To Read a Non-JSON Resource

With a mapping configured as described in "To Map an Alternative Content Type", REST client applications can read MIME resources as described in the following step:

- Read the non-JSON resource using a single value for each of the `_fields` and `_mimeType` parameters.

  The following example reads Babs Jensen's profile photo:

```
$ curl "http://kvaughan:bribery@opendj.example.com:8080/api/users/bjensen\
?_fields=photo&_mimeType=image/jpeg"
... binary data ...
```

Chapter 4

Performing LDAP Operations

In this chapter, you will learn how to use the command-line tools to perform LDAP operations.

About Command-Line Tools

Before you try the examples in this chatper, install command-line tools for performing LDAP operations. If you install server software, see "Server Command-Line Tools".

Client Command-Line Tools

This section covers the client tools installed with server software.

Before you try the examples in this guide, add client commands to your PATH:

```
# (UNIX)
$ export PATH=/path/to/opendj/bin:$PATH
```

```
# (Windows)
C:\>set PATH=\\path\to\opendj\bat:%PATH%
```

All DS command-line tools take the `--help` option.

All commands call Java programs and therefore involve starting a JVM.

The following list uses the UNIX names for the commands:

addrate

Measure add and delete throughput and response time.

For details, see "*addrate — measure add and delete throughput and response time*" in the *Reference*.

authrate

Measure bind throughput and response time.

For details, see "*authrate — measure bind throughput and response time*" in the *Reference*.

base64

Encode and decode data in base64 format.

Base64-encoding represents binary data in ASCII, and can be used to encode character strings in LDIF, for example.

For details, see "*base64 — encode and decode base64 strings*" in the *Reference*.

ldapcompare

Compare the attribute values you specify with those stored on entries in the directory.

For details, see "*ldapcompare — perform LDAP compare operations*" in the *Reference*.

ldapdelete

Delete entries from the directory.

For details, see "*ldapdelete — perform LDAP delete operations*" in the *Reference*.

ldapmodify

Modify the specified attribute values for the specified entries.

For details, see "*ldapmodify — perform LDAP modify, add, delete, mod DN operations*" in the *Reference*.

ldappasswordmodify

Modify user passwords.

For details, see "*ldappasswordmodify — perform LDAP password modifications*" in the *Reference*.

ldapsearch

Search a branch of directory data for entries that match the LDAP filter you specify.

For details, see "*ldapsearch — perform LDAP search operations*" in the *Reference*.

ldifdiff

Display differences between two LDIF files, with the resulting output having LDIF format.

For details, see "*ldifdiff — compare small LDIF files*" in the *Reference*.

ldifmodify

Similar to the **ldapmodify** command, modify specified attribute values for specified entries in an LDIF file.

For details, see "*ldifmodify — apply LDIF changes to LDIF*" in the *Reference*.

ldifsearch

Similar to the **ldapsearch** command, search a branch of data in LDIF for entries matching the LDAP filter you specify.

For details, see "*ldifsearch — search LDIF with LDAP filters*" in the *Reference*.

makeldif

Generate directory data in LDIF based on templates that define how the data should appear.

The **makeldif** command is designed to help generate test data that mimics data expected in production, but without compromising real, potentially private information.

For details, see "*makeldif — generate test LDIF*" in the *Reference*, and "*makeldif.template — template file for the makeldif command*" in the *Reference*.

modrate

Measure modification throughput and response time.

For details, see "*modrate — measure modification throughput and response time*" in the *Reference*.

searchrate

Measure search throughput and response time.

For details, see "*searchrate — measure search throughput and response time*" in the *Reference*.

Server Command-Line Tools

This section covers the tools installed with server software.

Before you try the examples in this guide, set your PATH to include the DS server tools. The location of the tools depends on the operating environment and on the packages used to install server software. "Paths To Administration Tools" indicates where to find the tools.

Paths To Administration Tools

DS running on...	DS installed from...	Default path to tools...
Linux distributions	.zip	`/path/to/opendj/bin`
Linux distributions	.deb, .rpm	`/opt/opendj/bin`
Microsoft Windows	.zip	`C:\path\to\opendj\bat`

You find the installation and upgrade tools, **setup**, and **upgrade**, in the parent directory of the other tools, as these tools are not used for everyday administration. For example, if the path to most tools

is `/path/to/opendj/bin` you can find these tools in `/path/to/opendj`. For instructions on how to use the installation and upgrade tools, see the Installation Guide.

All DS command-line tools take the `--help` option.

All commands call Java programs and therefore involve starting a JVM.

"Tools and Server Constraints" indicates the constraints, if any, that apply when using a command-line tool with a server.

Tools and Server Constraints

Commands	Constraints
backendstat **create-rc-script** **encode-password** **setup** **start-ds** **supportextract** **upgrade** **windows-service**	These commands must be used with the local DS server in the same installation as the tools. These commands are not useful with non-DS servers.
dsconfig **export-ldif** **import-ldif** **manage-account** **manage-tasks** **rebuild-index** **restore** **status** **stop-ds** **verify-index**	These commands must be used with DS servers having the same version as the command. These commands are not useful with non-DS servers.
dsreplication	• With one exception, this command can be used with current and previous DS server versions. The one exception is the **dsreplication reset-change-number** subcommand, which requires DS server version 3.0.0 or later. • This command does not support configuration expressions described in "Using Configuration Property Value Substitution" in the *Administration Guide*. When setting up and initializing replication in a DevOps environment, use the **dsreplication** command before substituting configuration values with expressions. • This command is not useful with other types of directory servers.

Commands	Constraints
makeldif	This command depends on template files. The template files can make use of configuration files installed with DS servers under `config/MakeLDIF/`. The LDIF output can be used with any directory server.
base64 **ldapcompare** **ldapdelete** **ldapmodify** **ldappasswordmodify** **ldapsearch** **ldifdiff** **ldifmodify** **ldifsearch**	These commands can be used independently of DS servers, and so are not tied to a specific version.

The following list uses the UNIX names for the commands. On Windows all command-line tools have the extension .bat:

addrate

Measure add and delete throughput and response time.

For details, see "*addrate — measure add and delete throughput and response time*" in the *Reference*.

authrate

Measure bind throughput and response time.

For details, see "*authrate — measure bind throughput and response time*" in the *Reference*.

backendstat

Debug databases for pluggable backends.

For details, see "*backendstat — gather OpenDJ backend debugging information*" in the *Reference*.

backup

Back up or schedule backup of directory data.

For details, see "*backup — back up directory data*" in the *Reference*.

base64

Encode and decode data in base64 format.

Base64-encoding represents binary data in ASCII, and can be used to encode character strings in LDIF, for example.

For details, see "*base64 — encode and decode base64 strings*" in the *Reference*.

changelogstat

Debug file-based changelog databases.

For details, see "*changelogstat — debug changelog and changenumber files*" in the *Reference*.

create-rc-script (UNIX)

Generate a script you can use to start, stop, and restart the server either directly or at system boot and shutdown. Use **create-rc-script -f *script-file***.

This allows you to register and manage DS servers as services on UNIX and Linux systems.

For details, see "*create-rc-script — script to manage OpenDJ as a service on UNIX*" in the *Reference*.

dsconfig

The **dsconfig** command is the primary command-line tool for viewing and editing DS server configurations. When started without arguments, **dsconfig** prompts you for administration connection information. Once connected to a running server, it presents you with a menu-driven interface to the server configuration.

To edit the configuration when the server is not running, use the `--offline` command.

Some advanced properties are not visible by default when you run the **dsconfig** command interactively. Use the `--advanced` option to access advanced properties.

When you pass connection information, subcommands, and additional options to **dsconfig**, the command runs in script mode and so is not interactive.

You can prepare **dsconfig** batch scripts by running the command with the `--commandFilePath` option in interactive mode, then reading from the batch file with the `--batchFilePath` option in script mode. Batch files can be useful when you have many **dsconfig** commands to run and want to avoid starting the JVM for each command.

Alternatively, you can read commands from standard input by using the `--batch` option.

For details, see "*dsconfig — manage OpenDJ server configuration*" in the *Reference*.

dsreplication

Configure data replication between directory servers to keep their contents in sync.

For details, see "*dsreplication — manage directory data replication*" in the *Reference*.

encode-password

Encode a cleartext password according to one of the available storage schemes.

For details, see "*encode-password — encode a password with a storage scheme*" in the *Reference*.

export-ldif

Export directory data to LDIF, the standard, portable, text-based representation of directory content.

For details, see "*export-ldif — export directory data in LDIF*" in the *Reference*.

import-ldif

Load LDIF content into the directory, overwriting existing data. It cannot be used to append data to the backend database.

For details, see "*import-ldif — import directory data from LDIF*" in the *Reference*.

ldapcompare

Compare the attribute values you specify with those stored on entries in the directory.

For details, see "*ldapcompare — perform LDAP compare operations*" in the *Reference*.

ldapdelete

Delete one entry or an entire branch of subordinate entries in the directory.

For details, see "*ldapdelete — perform LDAP delete operations*" in the *Reference*.

ldapmodify

Modify the specified attribute values for the specified entries.

For details, see "*ldapmodify — perform LDAP modify, add, delete, mod DN operations*" in the *Reference*.

ldappasswordmodify

Modify user passwords.

For details, see "*ldappasswordmodify — perform LDAP password modifications*" in the *Reference*.

ldapsearch

Search a branch of directory data for entries that match the LDAP filter you specify.

For details, see "*ldapsearch — perform LDAP search operations*" in the *Reference*.

ldifdiff

Display differences between two LDIF files, with the resulting output having LDIF format.

For details, see "*ldifdiff — compare small LDIF files*" in the *Reference*.

ldifmodify

Similar to the **ldapmodify** command, modify specified attribute values for specified entries in an LDIF file.

For details, see "*ldifmodify — apply LDIF changes to LDIF*" in the *Reference*.

ldifsearch

Similar to the **ldapsearch** command, search a branch of data in LDIF for entries matching the LDAP filter you specify.

For details, see "*ldifsearch — search LDIF with LDAP filters*" in the *Reference*.

makeldif

Generate directory data in LDIF based on templates that define how the data should appear.

The **makeldif** command is designed to help generate test data that mimics data expected in production, but without compromising real, potentially private information.

For details, see "*makeldif — generate test LDIF*" in the *Reference*.

manage-account

Lock and unlock user accounts, and view and manipulate password policy state information.

For details, see "*manage-account — manage state of OpenDJ server accounts*" in the *Reference*.

manage-tasks

View information about tasks scheduled to run in the server, and cancel specified tasks.

For details, see "*manage-tasks — manage server administration tasks*" in the *Reference*.

modrate

Measure modification throughput and response time.

For details, see "*modrate — measure modification throughput and response time*" in the *Reference*.

rebuild-index

Rebuild an index stored in an indexed backend.

For details, see "*rebuild-index — rebuild index after configuration change*" in the *Reference*.

restore

Restore data from backup.

For details, see "*restore — restore directory data backups*" in the *Reference*.

searchrate

Measure search throughput and response time.

For details, see "*searchrate — measure search throughput and response time*" in the *Reference*.

start-ds

Start one DS server.

For details, see "*start-ds — start OpenDJ server*" in the *Reference*.

status

Display information about the server.

For details, see "*status — display basic OpenDJ server information*" in the *Reference*.

stop-ds

Stop one DS server.

For details, see "*stop-ds — stop OpenDJ server*" in the *Reference*.

supportextract

Collect troubleshooting information for technical support purposes.

For details, see "*supportextract — extract support data*" in the *Reference*.

verify-index

Verify that an index stored in an indexed backend is not corrupt.

For details, see "*verify-index — check index for consistency or errors*" in the *Reference*.

windows-service (Windows)

Register and manage one DS server as a Windows Service.

For details, see "*windows-service — register DS as a Windows Service*" in the *Reference*.

How Command-Line Tools Trust Server Certificates

This section describes how DS command-line tools determine whether to trust server certificates.

When DS command-line tools connect securely to a server, the server presents its digital certificate. The tool must then determine whether to trust the server certificate and continue negotiating the secure connection, or not to trust the server certificate and drop the connection.

An important part of trusting a server certificate is trusting the signing certificate of the party who signed the server certificate. The process is described in more detail in "About Certificates, Private Keys, and Secret Keys" in the *Security Guide*.

Put simply, a tool can automatically trust the server certificate if the tool's truststore contains the signing certificate. "Command-Line Tools and Truststores" indicates where to find the truststore. The

signing certificate could be a CA certificate, or the server certificate itself if the certificate was self-signed.

When run in interactive mode, DS command-line tools can prompt you to decide whether to trust a server certificate not found in the truststore. If you have not specified a truststore and you choose to trust the certificate permanently, the tools store the certificate in a file. The file is `user.home/.opendj/keystore`, where *user.home* is the Java system property. *user.home* is `$HOME` on Linux and UNIX, and `%USERPROFILE%` on Windows. The keystore password is `OpenDJ`. Neither the file name nor the password can be changed.

When run in non-interactive mode, the tools either rely on the specified truststore, or use this default truststore if none is specified.

Command-Line Tools and Truststores

Truststore Option	Truststore Used
None	The default truststore, `$HOME/.opendj/keystore`, where `$HOME` is the home directory of the user running the command-line tool. When you choose interactively to permanently trust a server certificate, the certificate is stored in this truststore.

`-P {trustStorePath}`
 	
`--trustStorePath {trustStorePath}`
	Only the specified truststore is used.
	
	In this case, the tool does not allow you to choose interactively to permanently trust an unrecognized server certificate.

Searching the Directory

Searching the directory is akin to searching for a phone number in a paper phone book. You can look up a phone number because you know the last name of a subscriber's entry. In other words, you use the value of one attribute of the entry to find entries that have another attribute you want.

Whereas a paper phone book has only one index (alphabetical order by name), the directory has many indexes. When performing a search, you always specify which index to use, by specifying which attribute(s) you are using to lookup entries.

Your paper phone book might be divided into white pages for residential subscribers and yellow pages for businesses. If you are looking up an individual's phone number, you limit your search to the white pages. Directory services divide entries in various ways, often to separate organizations, and to separate groups from user entries from printers, for example, but potentially in other ways. When searching you therefore also specify where in the directory to search.

The **ldapsearch** command, described in "*ldapsearch — perform LDAP search operations*" in the *Reference*, thus takes at minimum a search base DN option and an LDAP filter. The search base DN

identifies where in the directory to search for entries that match the filter. For example, if you are looking for printers, you might specify the base DN as `ou=Printers,dc=example,dc=com`. Perhaps you are visiting the `GNB00` office and are looking for a printer as shown in the following example:

```
$ ldapsearch --baseDN ou=Printers,dc=example,dc=com "(printerLocation=GNB00)"
```

In the example, the LDAP filter indicates to the directory that you want to look up printer entries where the `printerLocation` attribute is equal to `GNB00`.

You also specify the host and port to access directory services, and the type of protocol to use (for example, LDAP/SSL, or StartTLS to protect communication). If the directory service does not allow anonymous access to the data you want to search, you also identify who is performing the search and provide their credentials, such as a password or certificate. Finally, you can specify a list of attributes to return. If you do not specify attributes, then the search returns all user attributes for the entry.

Review the following examples in this section to get a sense of how searches work:

- "Search: Using Simple Filters"
- "Search: Using Complex Filters"
- "Search: Return Operational Attributes"
- "Search: Returning Attributes for an Object Class"
- "Search: Finding an Approximate Match"
- "Search: Escaping Search Filter Characters"
- "Search: Listing Active Accounts"
- "Search: Performing a Persistent Search"
- "Search: Using Language Subtypes"
- "LDAP Filter Operators"
- "Search: Using JSON Query Filters"
- "Search: Using a JSON Assertion"
- "Search: Server-Side Sort"

For details about the operators that can be used in search filters, see "LDAP Filter Operators".

Search: Using Simple Filters

The following example searches for entries with user IDs (`uid`) containing `jensen`, returning only DNs and user ID values:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=*jensen*)" uid
dn: uid=ajensen,ou=People,dc=example,dc=com
uid: ajensen

dn: uid=bjensen,ou=People,dc=example,dc=com
uid: bjensen

dn: uid=gjensen,ou=People,dc=example,dc=com
uid: gjensen

dn: uid=jjensen,ou=People,dc=example,dc=com
uid: jjensen

dn: uid=kjensen,ou=People,dc=example,dc=com
uid: kjensen

dn: uid=rjensen,ou=People,dc=example,dc=com
uid: rjensen

dn: uid=tjensen,ou=People,dc=example,dc=com
uid: tjensen
```

Search: Using Complex Filters

The following example returns entries with `uid` containing `jensen` for users located in San Francisco:

```
$ ldapsearch \
 --port 1389 \
 --baseDN ou=people,dc=example,dc=com \
 "(&(uid=*jensen*)(l=San Francisco))" \
 @person
dn: uid=bjensen,ou=People,dc=example,dc=com
sn: Jensen
cn: Barbara Jensen
cn: Babs Jensen
objectClass: top
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: organizationalPerson
objectClass: person
description: Original description
telephoneNumber: +1 408 555 1862

dn: uid=rjensen,ou=People,dc=example,dc=com
sn: Jensen
cn: Richard Jensen
objectClass: top
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: organizationalPerson
objectClass: person
telephoneNumber: +1 408 555 5957
```

The command returns the attributes associated with the `person` object class.

Complex filters can use both "and" syntax, `(&(filtercomp)(filtercomp))`, and "or" syntax, `(|(filtercomp)(filtercomp))`.

Search: Return Operational Attributes

Use `+` in the attribute list after the filter to return all operational attributes, as in the following example:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=bjensen)" +
dn: uid=bjensen,ou=People,dc=example,dc=com
entryUUID: <uuid>
isMemberOf: cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com
subschemaSubentry: cn=schema
hasSubordinates: false
numSubordinates: 0
etag: <etag>
structuralObjectClass: inetOrgPerson
entryDN: uid=bjensen,ou=People,dc=example,dc=com
```

Alternatively, specify operational attributes by name.

Search: Returning Attributes for an Object Class

Use `@objectClass` in the attribute list after the filter to return the attributes associated with a particular object class as in the following example:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=bjensen)" @person
dn: uid=bjensen,ou=People,dc=example,dc=com
sn: Jensen
cn: Barbara Jensen
cn: Babs Jensen
objectClass: top
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: organizationalPerson
objectClass: person
description: Original description
telephoneNumber: +1 408 555 1862
```

Search: Finding an Approximate Match

DS servers support searches looking for an approximate match of the filter. Approximate match searches use the `~=` comparison operator, described in "LDAP Filter Operators". They rely on `approximate` type indexes, which are configured as shown in "Add an Approximate Index" in the *Administration Guide*.

The following example configures an approximate match index for the surname (`sn`) attribute, and then rebuilds the index:

```
$ dsconfig \
 set-backend-index-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name sn \
 --set index-type:approximate \
 --trustAll \
 --no-prompt
$ stop-ds --quiet
$ rebuild-index --offline --baseDN dc=example,dc=com --index sn
$ start-ds --quiet
```

Once the index is built, it is ready for use in searches. The following example shows a search using the approximate comparison operator:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(sn~=jansen)" sn
dn: uid=ajensen,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=bjense2,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=bjensen,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=ejohnson,ou=People,dc=example,dc=com
sn: Johnson

dn: uid=gjensen,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=jjensen,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=kjensen,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=rjense2,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=rjensen,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=tjensen,ou=People,dc=example,dc=com
sn: Jensen
```

Notice that `jansen` matches `Jensen` and `Johnson`, for example.

Search: Escaping Search Filter Characters

RFC 4515 *Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters*, mentions a number of characters that require special handing in search filters.

For a filter like `(attr=value)`, the following list indicates characters that you must replace with a backslash (`\`) followed by two hexadecimal digits when using them as part of the *value* string:

- Replace `*` with `\2a`.
- Replace `(` with `\28`.
- Replace `)` with `\29`.
- Replace `\` with `\5c`.
- Replace NUL (0x00) with `\00`.

The following example shows a filter with escaped characters matching an actual value:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(description=\28*\5c*\2a\29)" description
dn: uid=bjensen,ou=People,dc=example,dc=com
description: (A \great\ description*)
```

Search: Listing Active Accounts

DS servers support extensible matching rules, meaning you can pass in filters specifying a matching rule OID that extends your search beyond what you accomplish with standard LDAP.

DS servers support the generalized time-based matching rules described in "Configure an Extensible Match Index" in the *Administration Guide*:

- A partial date and time matching rule
- A greater-than relative time matching rule
- A less-than relative time matching rule

You can use these matching rules to list, for example, all users who have authenticated recently.

First set up an attribute to store a last login timestamp.

The following example defines a `lastLoginTime` attribute. Notice that the attribute has the property `USAGE directoryOperation`, meaning this is an operational attribute. When checking schema compliance, the server skips operational attributes. Operational attributes can therefore be added to an entry

without changing the entry's object classes. Furthermore, as described in "About Data In LDAP Directories", operational attributes hold information used by the directory itself. Operational attributes are only returned when explicitly requested, and not intended for modification by external applications. By defining the `lastLoginTime` attribute as operational, you limit its visibility and help prevent client applications from changing its value unless specifically granted access to do so.

You can do this by adding a schema definition for the attribute as in the following example:

```
$ cat lastLoginTime.ldif
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( lastLoginTime-oid
  NAME 'lastLoginTime'
  DESC 'Last time the user logged in'
  EQUALITY generalizedTimeMatch
  ORDERING generalizedTimeOrderingMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
  SINGLE-VALUE
  NO-USER-MODIFICATION
  USAGE directoryOperation
  X-ORIGIN 'DS example documentation' )

$ ldapmodify \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 lastLoginTime.ldif
```

Configure the applicable password policy to write the last login timestamp when a user authenticates. The following command configures the default password policy to write the timestamp in generalized time format to the `lastLoginTime` operational attribute on the user's entry:

```
$ dsconfig \
 set-password-policy-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --set last-login-time-attribute:lastLoginTime \
 --set last-login-time-format:"yyyyMMddHH'Z'" \
 --trustAll \
 --no-prompt
```

Configure and build an extensible matching rule index for time-based searches on the `lastLoginTime` attribute:

```
$ dsconfig \
 create-backend-index \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name dsEvaluation \
 --set index-type:extensible \
 --set index-extensible-matching-rule:1.3.6.1.4.1.26027.1.4.5 \
 --set index-extensible-matching-rule:1.3.6.1.4.1.26027.1.4.6 \
 --set index-extensible-matching-rule:1.3.6.1.4.1.26027.1.4.7 \
 --index-name lastLoginTime \
 --trustAll \
 --no-prompt
$ stop-ds --quiet
$ rebuild-index --offline --baseDN dc=example,dc=com --index lastLoginTime
$ start-ds --quiet
```

Make sure you have some users who have authenticated recently:

```
$ ldapsearch \
 --port 1389 \
 --bindDN uid=bjensen,ou=people,dc=example,dc=com \
 --bindPassword hifalutin \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 1.1

$ ldapsearch \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 1.1
```

The following search returns users who have authenticated in the last three months (13 weeks) according to the last login timestamps:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(lastLoginTime:1.3.6.1.4.1.26027.1.4.6:=13w)" \
 1.1
dn: uid=bjensen,ou=People,dc=example,dc=com

dn: uid=kvaughan,ou=People,dc=example,dc=com
```

The following search returns users who have authenticated in this year according to the last login timestamps:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(lastLoginTime:1.3.6.1.4.1.26027.1.4.7:=$(date +%Y))" \
 1.1
dn: uid=bjensen,ou=People,dc=example,dc=com

dn: uid=kvaughan,ou=People,dc=example,dc=com
```

Search: Performing a Persistent Search

DS servers and other LDAP servers support the Internet-Draft for *Persistent Search: A Simple LDAP Change Notification Mechanism*. A persistent search is like a search that never stops. Every time there is a change to an entry matching the search criteria, the search returns an additional response. Applications can also get change notifications by using a server's external change log as described in "Change Notification For Your Applications" in the *Administration Guide*.

In order to use the persistent search control with DS servers, the user performing the search must be given access to use the control. Persistent searches consume server resources, so directory administrators often limit permission to perform persistent searches to specific applications. If the user does not have access to use the control, the request to use the control causes the search operation to fail with a message such as the following:

```
The LDAP search request failed: 12 (Unavailable Critical Extension)
Additional Information:  The request control with Object Identifier (OID)
"2.16.840.1.113730.3.4.3" cannot be used due to insufficient access rights
```

An example of the ACI required is shown in "ACI Required For LDAP Operations" in the *Administration Guide*. The following command adds the permission for `My App` to perform persistent searches under `dc=example,dc=com`:

```
$ cat allow-psearch.ldif
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetcontrol = "2.16.840.1.113730.3.4.3")
 (version 3.0;acl "Allow Persistent Search for My App";
 allow (read)(userdn = "ldap:///cn=My App,ou=Apps,dc=example,dc=com");)

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 allow-psearch.ldif
```

To perform a persistent search, use the persistent search control, and optionally specify the type of changes for which to receive notifications, whether the server should return existing entries as well as changes, and whether to return additional entry change information with each notification. The additional entry change information returned is that of the entry change notification response control defined in the Internet-Draft. The response control indicates what type of change led to the notification, what the previous DN was if the change was a modify DN operation, and the change number if the LDAP server supports change numbers. For details about the options, see the description for the `--persistentSearch` option in "*ldapsearch — perform LDAP search operations*" in the *Reference*.

The following example initiates a persistent search, indicating that notifications should be sent for all update operations, only notifications about changed entries should be returned, and no additional information should be returned:

```
$ ldapsearch \
 --port 1389 \
 --bindDN 'cn=My App,ou=Apps,dc=example,dc=com' \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 --persistentSearch ps:all:true:false \
 '(&)' >> psearch.txt &
$ export PSEARCH_PID=$!
```

Notice the search filter, `(&)`, which is always true, meaning that it matches all entries.

The following modify and delete operations:

```
$ cat bjensen-psearch-description.ldif
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Hello, persistent search

$ ldapmodify \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 bjensen-psearch-description.ldif
$ ldapdelete \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 uid=tpierce,ou=People,dc=example,dc=com
```

Result in the following responses to the persistent search:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: posixAccount
objectClass: top
objectClass: organizationalPerson
objectClass: person
```

```
objectClass: inetOrgPerson
mail: bjensen@example.com
roomNumber: 0209
preferredLanguage: en, ko;q=0.8
manager: uid=trigden, ou=People, dc=example,dc=com
ou: Product Development
ou: People
givenName: Barbara
telephoneNumber: +1 408 555 1862
sn: Jensen
cn: Barbara Jensen
cn: Babs Jensen
homeDirectory: /home/bjensen
facsimileTelephoneNumber: +1 408 555 1992
gidNumber: 1000
userPassword: {SSHA512}<hash>
uidNumber: 1076
description: Hello, persistent search
uid: bjensen
l: San Francisco

dn: uid=tpierce,ou=People,dc=example,dc=com
objectClass: top
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: organizationalPerson
objectClass: person
mail: tpierce@example.com
roomNumber: 1383
manager: uid=scarter, ou=People, dc=example,dc=com
ou: Accounting
ou: People
givenName: Tobias
telephoneNumber: +1 408 555 1531
sn: Pierce
cn: Tobias Pierce
homeDirectory: /home/tpierce
facsimileTelephoneNumber: +1 408 555 9332
gidNumber: 1000
userPassword: {SSHA512}<hash>
uidNumber: 1042
uid: tpierce
l: Bristol
departmentNumber: 1000
preferredLanguage: en-gb
street: Broad Quay House, Prince Street
```

If the server is replicated, the output also includes the entry `dc=example,dc=com`. This is because the replication-related `ds-sync-*` operational attributes are updated, too. The entry therefore shows up in the persistent search results.

To terminate the persistent search, interrupt the command with **CTRL+C** (`SIGINT`) or `SIGTERM` as in the following command:

```
$ kill -s SIGTERM $PSEARCH_PID
```

Search: Using Language Subtypes

DS servers support many language subtypes. For a list, see "*Localization*" in the *Reference*.

When you perform a search you can request the language subtype by OID or by language subtype string. For example, the following search gets the French version of a common name. The example uses the DS **base64** command to decode the attribute value:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 "(givenName:fr:=Fréderique)" cn\;lang-fr
dn: uid=fdupont,ou=People,dc=example,dc=com
cn;lang-fr:: RnJlZMOpcmlxdWUgRHVwb250
$ base64 decode --encodedData RnJlZMOpcmlxdWUgRHVwb250
Frédérique Dupont
```

At the end of the OID or language subtype, further specify the matching rule as follows:

- Add `.1` for less than
- Add `.2` for less than or equal to
- Add `.3` for equal to (default)
- Add `.4` for greater than or equal to
- Add `.5` for greater than
- Add `.6` for substring

The following table describes the operators you can use in LDAP search filters.

LDAP Filter Operators

Operator	Definition	Example
`=`	Equality comparison, as in `(sn=Jensen)`. This can also be used with substring matches. For example, to match last names starting with `Jen`, use the filter `(sn=Jen*)`. Substrings are more expensive for the directory server to index. Substring searches therefore might not be permitted for many attributes.	`"(cn=My App)"` matches entries with common name `My App`. `"(sn=Jen*)"` matches entries with surname starting with `Jen`.
`<=`	Less than or equal to comparison, which works alphanumerically.	`"(cn<=App)"` matches entries with `commonName` up to those starting with App (case-insensitive) in alphabetical order.

Operator	Definition	Example
`>=`	Greater than or equal to comparison, which works alphanumerically.	`"(uidNumber>=1151)"` matches entries with `uidNumber` greater than 1151.
`=*`	Presence comparison. For example, to match all entries having a `userPassword`, use the filter `(userPassword=*)`.	`"(member=*)"` matches entries with a `member` attribute.
`~=`	Approximate comparison, matching attribute values similar to the value you specify.	`"(sn~=jansen)"` matches entries with a surname that sounds similar to `Jansen` (Johnson, Jensen, and other surnames).
`[:dn][:oid]:=`	Extensible match comparison. At the end of the OID or language subtype, you further specify the matching rule as follows: • Add `.1` for less than • Add `.2` for less than or equal to • Add `.3` for equal to (default) • Add `.4` for greater than or equal to • Add `.5` for greater than • Add `.6` for substring	`(uid:dn:=bjensen)` matches entries where `uid` having the value `bjensen` is a component of the entry DN. `(lastLoginTime: 1.3.6.1.4.1.26027.1.4.5:=-13w)` matches entries with a last login time more recent than 13 weeks. You also use extensible match filters with localized values. Directory services like DS support a variety of internationalized locales, each of which has an OID for collation order, such as `1.3.6.1.4.1.42.2.27.9.4.76.1` for French. DS software lets you use the language subtype, such as `fr`, instead of the OID. `"(cn:dn:=My App)"` matches entries who have `My App` as the common name and also as the value of a DN component.
`!`	NOT operator, to find entries that do not match the specified filter component. Take care to limit your search when using `!` to avoid matching so many entries that the server treats your search as unindexed.	`'!(objectclass=person)'` matches non-person entries.
`&`	AND operator, to find entries that match all specified filter components.	`'(&(l=San Francisco)(!(uid=bjensen)))'` matches entries for users in San Francisco other than the user with ID `bjensen`.
`|`	OR operator, to find entries that match one of the specified filter components.	`"|(sn=Jensen)(sn=Johnson)"` matches entries with surname Jensen or surname Johnson.

Search: Using JSON Query Filters

DS servers support attribute values that have JSON syntax. This makes it possible to index JSON values, and to search for them using Common REST query filters, as described in "Query".

This example depends on the configuration described in "Example: Custom Index Using a JSON Query Matching Rule" in the *Administration Guide*. The settings are applied when you set up a

directory server for evaluation as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*. Search for entries with a `json` attribute:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(json=*)" json
dn: uid=bjensen,ou=People,dc=example,dc=com
json: {"access_token":"123","expires_in":59,"token_type":"Bearer","refresh_token":"456"}

dn: uid=scarter,ou=People,dc=example,dc=com
json: {"access_token":"789","expires_in":59,"token_type":"Bearer","refresh_token":"012"}
```

Notice that you can search with Common REST query filters. The following example finds the entry with a JSON attribute whose `access_token` field has a value of `123`:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(json=access_token eq '123')" json
dn: uid=bjensen,ou=People,dc=example,dc=com
json: {"access_token":"123","expires_in":59,"token_type":"Bearer","refresh_token":"456"}
```

Be aware that you can combine Common REST query filter syntax filters with other LDAP search filter to form complex filters as demonstrated in "Search: Using Complex Filters". For example, `(&(json=access_token eq '123')(mail=bjensen@example.com))`.

Search: Using a JSON Assertion

In addition to searches with query filters, JSON attributes can be matched with filters using JSON in the assertion. This example demonstrates a case where JSON objects are considered equal if their "id" fields match.

This example depends on the configuration described in "Example: Index Using a Custom JSON Equality Matching Rule" in the *Administration Guide*. The settings are applied when you set up a directory server for evaluation as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*. Search for entries with a `jsonToken` attribute:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com '(jsonToken={"id":"HgAaB6xDhLom4JbM"})' jsonToken
jsonToken: {"id":"HgAaB6xDhLom4JbM","scopes":["read","write"],"expires":"2018-01-10T10:08:34Z"}
```

Search: Server-Side Sort

If permitted by the directory administrator, you can request that the server sort the search results before returning them. When your application requests a server-side sort, the server retrieves the entries matching your search, and then returns the whole set of entries in sorted order. This process consumes memory resources on the server, so the best practice is to sort the results after you retrieve them, or to browse results with a search that matches a virtual list view index as demonstrated in "Configuring a Virtual List View Index" in the *Administration Guide*.

This example demonstrates a server-side sort request, where the results are sorted by surname:

```
$ cat allow-server-side-sort.ldif
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetcontrol = "2.16.840.1.113730.3.4.3")
 (version 3.0;acl "Allow Server-Side Sort for Kirsten Vaughan";
 allow (read)(userdn = "ldap:///uid=kvaughan,ou=People,dc=example,dc=com");)

$ ldapmodify \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDn "cn=Directory Manager" \
 --bindPassword password \
 allow-server-side-sort.ldif
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDn uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDn dc=example,dc=com \
 --sortOrder +sn \
 "(sn=jensen)" \
 givenName sn
dn: uid=ajensen,ou=People,dc=example,dc=com
givenName: Allison
sn: Jensen

dn: uid=bjense2,ou=People,dc=example,dc=com
givenName: Bjorn
sn: Jensen

dn: uid=bjensen,ou=People,dc=example,dc=com
givenName: Barbara
sn: Jensen

dn: uid=gjensen,ou=People,dc=example,dc=com
givenName: Gern
sn: Jensen

dn: uid=jjensen,ou=People,dc=example,dc=com
givenName: Jody
sn: Jensen

dn: uid=kjensen,ou=People,dc=example,dc=com
givenName: Kurt
sn: Jensen

dn: uid=rjense2,ou=People,dc=example,dc=com
givenName: Randy
sn: Jensen

dn: uid=rjensen,ou=People,dc=example,dc=com
givenName: Richard
sn: Jensen

dn: uid=tjensen,ou=People,dc=example,dc=com
```

```
givenName: Ted
sn: Jensen
```

For use with JSON attributes, DS servers extend the standard `[+|-]attr` sort order syntax to sort on fields inside JSON objects.

The extended syntax is `[+|-]ldapAttr[:extensibleJsonOrderingMatchingRule:caseSensitive:ignoreWhiteSpace:/jsonPath[:/jsonPath ...]]`, where:

All arguments are mandatory when using the extended form:

- *extensibleJsonOrderingMatchingRule* is a JSON ordering matching rule name or OID.

  If you plan to create your own ordering indexes based on the matching rule, then first define it in the LDAP schema and create a custom schema provider for the matching rule. For details, see "Working With JSON" in the *Administration Guide*.

  Otherwise, if the JSON ordering matching rule is not yet implemented, the DS server creates it on demand to process the request. This makes it possible to sort on any field of a JSON attribute value, although the search is unindexed.

- *caseSensitive* is a boolean.

  Set to `true` to respect case when comparing values, `false` otherwise.

- *ignoreWhiteSpace* is a boolean.

  Set to `true` to ignore whitespace when comparing values, `false` otherwise.

- Each *jsonPath* is a path to a field inside the JSON object.

  Specify at least one *jsonPath*.

Comparing Attribute Values

The compare operation checks whether an attribute value you specify matches the attribute value stored on one or more directory entries.

Compare: Checking `authPassword`

In this example, Kirsten Vaughan uses the **ldapcompare** command, described in "*ldapsearch — perform LDAP search operations*" in the *Reference*, to check whether the hashed password value matches the stored value on `authPassword`:

```
$ ldapcompare \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 'authPassword:MD5$dFHgpDxXUT8=$qlC4xMXvmVlusJLz9/WJ5Q==' \
 uid=kvaughan,ou=people,dc=example,dc=com
# Comparing type authPassword with value MD5$dFHgpDxXUT8=$qlC4xMXvmVlusJLz9/WJ5Q== in entry
 uid=kvaughan,ou=people,dc=example,dc=com
# Compare operation returned true for entry uid=kvaughan,ou=people,dc=example,dc=com
```

Updating the Directory

Authorized users can change directory data using the LDAP add, modify, modify DN, and delete operations. You can use the **ldapmodify** command to make changes. For details see "*ldapmodify — perform LDAP modify, add, delete, mod DN operations*" in the *Reference*.

Adding Entries

With the **ldapmodify** command, authorized users can add entire entries from LDIF.

This section includes the following examples:

- "Adding Two New Users"
- "Bulk Adding Entries"

Adding Two New Users

The following example adds two new users:

```
$ cat new-users.ldif
dn: cn=Arsene Lupin,ou=Special Users,dc=example,dc=com
objectClass: person
objectClass: top
cn: Arsene Lupin
telephoneNumber: +33 1 23 45 67 89
sn: Lupin

dn: cn=Horace Velmont,ou=Special Users,dc=example,dc=com
objectClass: person
objectClass: top
cn: Horace Velmont
telephoneNumber: +33 1 12 23 34 45
sn: Velmont

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 new-users.ldif
```

Bulk Adding Entries

The following example adds ten thousand generated entries, using the `--numConnections` option to perform multiple add operations in parallel:

```
$ # Generate user entries, removing container entries from the output:
makeldif \
 --outputLdif output.ldif \
 /path/to/opendj/config/MakeLDIF/example.template ; \
 sed '1,10d' output.ldif > generated-users.ldif

$ # Bulk add the generated user entries:
ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --numConnections 64 \
 generated-users.ldif
```

When you use the `--numConnections` option, the number of connection is rounded up to the nearest power of two for performance reasons.

Modifying Entry Attributes

With the **ldapmodify** command, authorized users can change the values of attributes in the directory using LDIF as specified in RFC 2849.

This section includes the following examples:

- "Modify: Adding Attributes"
- "Modify: Changing an Attribute Value"
- "Modify: Deleting an Attribute"
- "Modify: Deleting a Single Attribute Value"
- "Modify: Applying Changes From Standard Input"
- "Modify: Using Optimistic Concurrency"
- "Modify: Updating a JSON Syntax Attribute"

Modify: Adding Attributes

The following example shows you how to add a description and JPEG photo to Sam Carter's entry:

```
$ cat scarter-mods.ldif
dn: uid=scarter,ou=people,dc=example,dc=com
changetype: modify
add: description
description: Accounting Manager
-
add: jpegphoto
jpegphoto:<file:///tmp/picture.jpg

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 scarter-mods.ldif
```

Modify: Changing an Attribute Value

The following example replaces the description on Sam Carter's entry:

```
$ cat scarter-newdesc.ldif
dn: uid=scarter,ou=people,dc=example,dc=com
changetype: modify
replace: description
description: New description

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 scarter-newdesc.ldif
```

Modify: Deleting an Attribute

The following example deletes the JPEG photo on Sam Carter's entry:

```
$ cat scarter-delphoto.ldif
dn: uid=scarter,ou=people,dc=example,dc=com
changetype: modify
delete: jpegphoto

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 scarter-delphoto.ldif
```

Modify: Deleting a Single Attribute Value

The following example deletes a single CN value on Barbara Jensen's entry:

```
$ cat delete-cn.ldif
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
delete: cn
cn: Barbara Jensen

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 delete-cn.ldif
```

Modify: Applying Changes From Standard Input

A double dash, `--`, signifies the end of command options, after which only trailing arguments are allowed. To indicate standard input as a trailing argument, use a bare dash, `-`, after the double dash.

Consider the following changes expressed in LDIF:

```
$ cat bjensen-stdin-description.ldif
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: New description from standard input
```

To send these changes to the **ldapmodify** command on standard input, use either of the following equivalent constructions:

```
# With dashes:
$ cat bjensen-stdin-description.ldif | ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 -- -
# Without dashes:
$ cat bjensen-stdin-description.ldif | ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery
```

Modify: Using Optimistic Concurrency

Imagine you are writing an application that lets end users update user profiles through a browser. You store user profiles as DS entries. Your end users can look up user profiles and modify them. Your application assumes that the end users can tell the right information when they see it, and updates profiles exactly as users see them on their screens.

Consider two users, Alice and Bob, both busy and often interrupted. Alice has Babs Jensen's new phone and room numbers. Bob has Babs's new location and description. Both assume that they have all the information that has changed. What can you do to make sure that your application applies the right changes when Alice and Bob simulaneously update Babs Jensen's profile?

DS servers have two features to help you in this situation. One of the features is the LDAP Assertion Control, described in Assertion request control in the *Reference*, used to tell the directory server to perform the modification only if an assertion you make stays true. The other feature is DS support for entity tag (ETag) attributes, making it easy to check whether the entry in the directory is the same as the entry you read.

Alice and Bob both get Babs's entry. In LDIF, the relevant attributes from the entry look like this. The ETag is a generated value that depends on the content of the entry:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 telephoneNumber roomNumber l ETag
dn: uid=bjensen,ou=People,dc=example,dc=com
telephoneNumber: +1 408 555 1862
roomNumber: 0209
l: San Francisco
ETag: ETAG
```

Bob prepares his changes in your application. Bob is almost ready to submit the new location and description when Carol stops by to ask Bob a few questions.

Alice starts just after Bob, but manages to submit her changes without interruption. Now Babs's entry has a new phone number, room number, and ETag:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 telephoneNumber roomNumber l ETag
dn: uid=bjensen,ou=People,dc=example,dc=com
telephoneNumber: +47 2108 1746
roomNumber: 1389
l: San Francisco
ETag: NEW_ETAG
```

In your application, you use the ETag value with the assertion control to prevent Bob's update from succeeding although the entry has changed. Your application tries the equivalent of the following commands with Bob's updates:

```
$ cat bjensen-stdin-description.ldif
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: l
l: Grenoble
-
add: description
description: Employee of the Month

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --assertionFilter "(ETag=${ETAG})" \
 alices.ldif
# The LDAP modify request failed: 122 (Assertion Failed)
# Additional Information:  Entry uid=bjensen,ou=People,dc=example,dc=com cannot be modified because the
 request contained an LDAP assertion control and the associated filter did not match the contents of the
 entry
```

Your application reloads Babs's entry with the new ETag value, and tries Bob's update again. This time Bob's changes do not collide with other changes. Babs's entry is successfully updated:

```
$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --assertionFilter "(ETag=${NEW_ETAG})" \
 alices.ldif
# MODIFY operation successful for DN uid=bjensen,ou=People,dc=example,dc=com
```

Modify: Updating a JSON Syntax Attribute

DS servers support attribute values that have JSON syntax as demonstrated in "Search: Using JSON Query Filters".

This example depends on the configuration and sample data used in "Example: Custom Index Using a JSON Query Matching Rule" in the *Administration Guide*. Perform the commands in that example to prepare the server before trying this one.

The following example replaces the existing JSON value with a new JSON value:

```
$ cat json-mod.ldif
dn: uid=bjensen,ou=people,dc=example,dc=com
changetype: modify
replace: json
json: {"stuff":["things","devices","paraphernalia"]}

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 json-mod.ldif
```

Notice that the JSON object is replaced entirely.

When DS servers receive update requests for `Json` syntax attributes, they expect valid JSON objects. By default, `Json` syntax attribute values must comply with *The JavaScript Object Notation (JSON) Data Interchange Format* described in RFC 7159. You can use the advanced core schema configuration option `json-validation-policy` to have the server be more lenient in what it accepts, or to disable JSON syntax checking.

The following example relaxes JSON syntax checking to allow comments, single quotes, and unquoted control characters, such as newlines, in strings:

```
$ dsconfig \
 set-schema-provider-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --provider-name "Core Schema" \
 --set json-validation-policy:lenient \
 --trustAll \
 --no-prompt
```

Filtering Add and Modify Operations

Some client applications send updates including attributes with names that differ from the attribute names defined in the LDAP schema. Other client applications might try to update attributes they should not update, such as the operational attributes `creatorsName`, `createTimestamp`, `modifiersName`, and `modifyTimestamp`. Ideally, you would fix the client application behavior, but that is not always feasible.

You can configure the attribute cleanup plugin to filter add and modify requests, rename attributes in requests using incorrect names, and remove attributes that applications should not change.

Renaming Incoming Attributes

The following example renames incoming `email` attributes to `mail` attributes. First, configure the attribute cleanup plugin to rename the inbound attribute:

```
$ dsconfig \
 create-plugin \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --type attribute-cleanup \
 --plugin-name "Rename email to mail" \
 --set enabled:true \
 --set rename-inbound-attributes:email:mail \
 --trustAll \
 --no-prompt
```

Next, confirm that it worked as expected:

```
$ cat email.ldif
dn: uid=newuser,ou=People,dc=example,dc=com
uid: newuser
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
cn: New User
sn: User
ou: People
email: newuser@example.com
userPassword: changeme

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 email.ldif
$ ldapsearch --port 1389 --baseDN dc=example,dc=com uid=newuser mail
dn: uid=newuser,ou=People,dc=example,dc=com
mail: newuser@example.com
```

Removing Incoming Attributes

The following example prevents client applications from adding or modifying `creatorsName`, `createTimestamp`, `modifiersName`, and `modifyTimestamp` attributes. First, set up the attribute cleanup plugin:

```
$ dsconfig \
 create-plugin \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --type attribute-cleanup \
 --plugin-name "Remove attrs" \
 --set enabled:true \
 --set remove-inbound-attributes:creatorsName \
 --set remove-inbound-attributes:createTimestamp \
 --set remove-inbound-attributes:modifiersName \
 --set remove-inbound-attributes:modifyTimestamp \
 --trustAll \
 --no-prompt
```

Next, confirm that it worked as expected:

```
$ cat badattrs.ldif
dn: uid=badattr,ou=People,dc=example,dc=com
uid: newuser
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
cn: Bad Attr
sn: Attr
ou: People
mail: badattr@example.com
userPassword: changeme
creatorsName: cn=Bad Attr
createTimestamp: Never in a million years.
modifiersName: cn=Directory Manager
modifyTimestamp: 20110930164937Z

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 badattrs.ldif
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 "(uid=badattr)" \
 creatorsName createTimestamp modifiersTimestamp modifyTimestamp
dn: uid=badattr,ou=People,dc=example,dc=com
creatorsName: uid=kvaughan,ou=people,dc=example,dc=com
createTimestamp: <timestamp>
```

Renaming Entries

The Relative Distinguished Name (RDN) refers to the part of an entry's DN that differentiates it from all other DNs at the same level in the directory tree. For example, `uid=bjensen` is the RDN of the entry with the DN `uid=bjensen,ou=People,dc=example,dc=com`.

With the **ldapmodify** command, authorized users can rename entries in the directory.

When you change the RDN of the entry, you are renaming the entry, modifying the value of the naming attribute, and the entry's DN.

Rename: Modifying the DN

Sam Carter is changing her last name to Jensen, and changing her login from `scarter` to `sjensen`. The following example shows you how to rename and change Sam Carter's entry. Notice the boolean field, `deleteoldrdn: 1`, which indicates that the previous RDN, `uid: scarter`, should be removed. (Setting `deleteoldrdn: 0` instead would preserve `uid: scarter` on the entry.)

```
$ cat scarter-sjensen.ldif
dn: uid=scarter,ou=people,dc=example,dc=com
changetype: modrdn
newrdn: uid=sjensen
deleteoldrdn: 1

dn: uid=sjensen,ou=people,dc=example,dc=com
changetype: modify
replace: cn
cn: Sam Jensen
-
replace: sn
sn: Jensen
-
replace: homeDirectory
homeDirectory: /home/sjensen
-
replace: mail
mail: sjensen@example.com

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 scarter-sjensen.ldif
```

Moving Entries

When you rename an entry with child entries, the directory has to move all the entries underneath it.

Note

DS directory servers support the modify DN operation only for moving entries in the same backend, under the same suffix. Depending on the number of entries you move, this can be a resource-intensive operation.

Moving an Entire Branch in the Same Backend

The following example moves all entries at and below `ou=People,dc=example,dc=com` under `ou=Subscribers,dc=example,dc=com`. All the entries in this example are in the same backend. The line `deleteoldrdn: 1` indicates that the old RDN, `ou: People`, should be removed:

```
$ cat move-ou-people.ldif
dn: ou=People,dc=example,dc=com
changetype: modrdn
newrdn: ou=Subscribers
deleteoldrdn: 1
newsuperior: dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 move-ou-people.ldif
```

Be aware that the move does not modify ACIs and other values that depend on `ou=People`. You must also edit any affected entries.

Moving One Entry at a Time

The following example moves an application entry that is under `dc=example,dc=com` under `ou=Apps,dc=example,dc=com` instead. The line `deleteoldrdn: 0` indicates that old RDN, `cn`, should be preserved:

```
$ cat move-app.ldif
dn: cn=New App,dc=example,dc=com
changetype: moddn
newrdn: cn=An App
deleteoldrdn: 0
newsuperior: ou=Apps,dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 move-app.ldif
```

Deleting Entries

With the **ldapdelete** command, authorized users can delete entries from the directory.

This section includes the following examples:

- "Delete: Removing a Subtree"
- "Delete: Applying a List From Standard Input"

Delete: Removing a Subtree

The following example shows you how to grant access to use the subtree delete control to an administrator, and to use the subtree delete option to remove an entry and its child entries:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --add global-aci:"(targetcontrol=\"1.2.840.113556.1.4.805\")\
 (version 3.0; acl \"Allow Subtree Delete\"; allow(read) \
 userdn=\"ldap:///uid=kvaughan,ou=People,dc=example,dc=com\";)" \
 --trustAll \
 --no-prompt
$ ldapdelete \
--port 1389 \
--bindDN "uid=kvaughan,ou=People,dc=example,dc=com" \
--bindPassword bribery \
--deleteSubtree "ou=Special Users,dc=example,dc=com"
```

Delete: Applying a List From Standard Input

A double dash, `--`, signifies the end of command options, after which only trailing arguments are allowed. To indicate standard input as a trailing argument, use a bare dash, `-`, after the double dash.

Consider the following list of users to delete:

```
$ cat users-to-delete.txt
uid=sfarmer,ou=People,dc=example,dc=com
uid=skellehe,ou=People,dc=example,dc=com
uid=slee,ou=People,dc=example,dc=com
uid=smason,ou=People,dc=example,dc=com
uid=speterso,ou=People,dc=example,dc=com
uid=striplet,ou=People,dc=example,dc=com
```

To send this list to the **ldapdelete** command on standard input, use either of the following equivalent constructions:

```
# With dashes:
$ cat users-to-delete.txt | ldapdelete \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 -- -
# Without dashes:
$ cat users-to-delete.txt | ldapdelete \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery
```

Changing Passwords

With the **ldappasswordmodify** command, described in "*ldappasswordmodify — perform LDAP password modifications*" in the *Reference*, authorized users can change and reset user passwords.

Resetting Passwords

As a password administrator, Kirsten Vaughan has been given the `password-reset` privilege as shown in "To Add Privileges for an Individual Administrator" in the *Administration Guide*. The following example shows Kirsten resetting Andy Hall's password.

```
$ ldappasswordmodify \
 --useStartTLS \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --authzID "dn:uid=ahall,ou=people,dc=example,dc=com" \
 --trustAll
The LDAP password modify operation was successful
Generated Password:  <password>
```

Tip

The **ldappasswordmodify** command uses the LDAP Password Modify extended operation. If this extended operation is performed on a connection that is already associated with a user (in other words, when a user first does a bind on the connection, then requests the LDAP Password Modify extended operation), then the operation is performed as the user associated with the connection. If the user associated with the connection is not the same user whose password is being changed, then DS servers consider it a password reset.

Whenever one user changes another user's password, DS servers consider it a password reset. Often password policies specify that users must change their passwords again after a password reset.

If you want your application to change a user's password, rather than reset a user's password, have your application request the password change as the user whose password is changing.

To change the password as the user, bind as the user whose password should be changed, and use the LDAP Password Modify extended operation with an authorization ID but without performing a bind, or use proxied authorization. For instructions on using proxied authorization, see "Configuring Proxied Authorization".

You can adjust the setting for password reset with the **manage-account** command, described in "*manage-account — manage state of OpenDJ server accounts*" in the *Reference*, The **set-password-is-reset** option is a hidden option, supported only for testing:

```
$ manage-account \
 set-password-is-reset \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --targetDN uid=ahall,ou=people,dc=example,dc=com \
 --operationValue true \
 --trustAll
```

Changing One's Own Password

Users can change their own passwords with the **ldappasswordmodify** command as long as they know their current password:

```
$ ldappasswordmodify \
 --useStartTLS \
 --port 1389 \
 --authzID "dn:uid=ahunter,ou=people,dc=example,dc=com" \
 --currentPassword egregious \
 --newPassword secret12 \
 --trustAll
```

The same operation works for directory superusers, such as `cn=Directory Manager`:

```
$ ldappasswordmodify \
 --useStartTLS \
 --port 1389 \
 --authzID "dn:cn=Directory Manager" \
 --currentPassword password \
 --newPassword secret12 \
 --trustAll
```

Changing Passwords With Special Characters

DS servers expect passwords to be UTF-8 encoded (base64-encoded when included in LDIF):

```
$ export LANG=en_US.UTF-8
$ ldappasswordmodify \
 --useStartTLS \
 --port 1389 \
 --bindDN uid=wlutz,ou=People,dc=example,dc=com \
 --bindPassword bassinet \
 --currentPassword bassinet \
 --newPassword pàsswôrd \
 --trustAll
$ ldapsearch \
 --port 1389 \
 --bindDN uid=wlutz,ou=People,dc=example,dc=com \
 --bindPassword pàsswôrd \
 --baseDN dc=example,dc=com \
 "(uid=wlutz)" \
 1.1
dn: uid=wlutz,ou=People,dc=example,dc=com
```

Configuring Default Settings

You can use `~/.opendj/tools.properties` to set the defaults for bind DN, host name, and port number as in the following example:

```
hostname=directory.example.com
port=1389
bindDN=uid=kvaughan,ou=People,dc=example,dc=com

ldapcompare.port=1389
ldapdelete.port=1389
ldapmodify.port=1389
ldappasswordmodify.port=1389
ldapsearch.port=1389
```

The location on Windows is `%UserProfile%/.opendj/tools.properties`.

Authenticating To the Directory Server

Authentication is the act of confirming the identity of a principal. Authorization is the act of determining whether to grant or to deny access to a principal. Authentication is performed to make authorization decisions.

As explained in "*Configuring Privileges and Access Control*" in the *Administration Guide*, DS servers implement fine-grained access control for authorization. Authorization for an operation depends on who is requesting the operation. In LDAP, directory servers must therefore authenticate a principal before they can authorize or deny access for particular operations. In LDAP, the bind operation authenticates the principal. The first LDAP operation in every LDAP session is generally a bind.

Clients bind by providing both a means to find their principal's entry in the directory and also by providing some credentials that the directory server can check against their entry.

In the simplest bind operation, the client provides a zero-length name and a zero-length password. This results in an anonymous bind, meaning the client is authenticated as an anonymous user of the directory. In the simplest examples in "Searching the Directory", notice that no authentication information is provided. The examples work because the client commands default to requesting anonymous binds when no credentials are provided, and because access controls for the sample data allow anonymous clients to read, search, and compare some directory data.

In a simple bind operation, the client provides an LDAP name, such as the DN identifying its entry, and the corresponding password stored on the `userPassword` attribute of the entry. In "Updating the Directory", notice that to change directory data, the client provides the bind DN and bind password of a user who has permission to change directory data. The commands do not work with a bind DN and bind password because access controls for the sample data only let authorized users change directory data.

Users rarely provide client applications with DNs, however. Instead, users might provide a client application with an identity string like a user ID or an email address. Depending on how the DNs are constructed, the client application can either build the DN directly from the user's identity string, or use a session where the bind has been performed with some other identity to search for the user entry based on the user's identity string. Given the DN constructed or found, the client application can then perform a simple bind.

For example, suppose Babs Jensen enters her email address, `bjensen@example.com`, and her password in order to log in. The client application might search for the entry matching `(mail=bjensen@example.com)` under base DN `dc=example,dc=com`. Alternatively, the client application might know to extract the user ID `bjensen` from the address, then build the corresponding DN, `uid=bjensen,ou=people,dc=example,dc=com` in order to bind.

When an identifier string provided by the user can be readily mapped to the user's entry DN, DS servers can translate between the identifier string and the entry DN. This translation is the job of a component called an identity mapper. Identity mappers are used to perform PLAIN SASL authentication (with a user name and password), SASL GSSAPI authentication (Kerberos V5), SASL CRAM MD5, and DIGEST MD5 authentication. They also handle authorization IDs during password modify extended operations and proxied authorization.

One use of PLAIN SASL is to translate user names from HTTP Basic authentication to LDAP authentication. The following example shows PLAIN SASL authentication using the default Exact Match identity mapper. In this (contrived) example, Babs Jensen reads the hashed value of her password. (According to the access controls in the example data, Babs must authenticate to read her password.) Notice the authentication ID is her user ID, `u:bjensen`, rather than the DN of her entry:

```
$ ldapsearch \
 --port 1389 \
 --useStartTLS \
 --baseDN dc=example,dc=com \
 --saslOption mech=PLAIN \
 --saslOption authid=u:bjensen \
 --bindPassword hifalutin \
 --trustAll \
 "(cn=Babs Jensen)" \
 userPassword
dn: uid=bjensen,ou=People,dc=example,dc=com
userPassword: {SSHA512}<hash>
```

The Exact Match identity mapper searches for a match between the string provided (here, `bjensen`) and the value of a specified attribute (by default the `uid` attribute). If you know users are entering their email addresses, you could create an exact match identity mapper for email addresses, then use that for PLAIN SASL authentication as in the following example:

```
$ dsconfig \
 create-identity-mapper \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --mapper-name "Email Mapper" \
 --type exact-match \
 --set match-attribute:mail \
 --set enabled:true \
 --trustAll \
 --no-prompt
$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name PLAIN \
 --set identity-mapper:"Email Mapper" \
 --trustAll \
 --no-prompt
$ ldapsearch \
 --port 1389 \
 --useStartTLS \
 --baseDN dc=example,dc=com \
 --saslOption mech=PLAIN \
 --saslOption authid=u:bjensen@example.com \
 --bindPassword hifalutin \
 --trustAll \
 "(cn=Babs Jensen)" \
 userPassword
dn: uid=bjensen,ou=People,dc=example,dc=com
userPassword: {SSHA512}<hash>
```

The DS Regular Expression identity mapper uses a regular expression to extract a substring from the string provided, then searches for a match between the substring and the value of a specified attribute. In the case of example data where an email address is *user ID* + @ + *domain*, you can use the default Regular Expression identity mapper in the same way as the email mapper from the previous example. The default regular expression pattern is `^([^@]+)@.+$`, and the part of the identity string matching `([^@]+)` is used to find the entry by user ID:

```
$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name PLAIN \
 --set identity-mapper:"Regular Expression" \
 --trustAll \
 --no-prompt
$ ldapsearch \
 --port 1389 \
 --useStartTLS \
 --baseDN dc=example,dc=com \
 --saslOption mech=PLAIN \
 --saslOption authid=u:bjensen@example.com \
 --bindPassword hifalutin \
 --trustAll \
 "(cn=Babs Jensen)" \
 userPassword
dn: uid=bjensen,ou=People,dc=example,dc=com
userPassword: {SSHA512}<hash>
```

Try the **dsconfig** command interactively to experiment with `match-pattern` and `replace-pattern` settings for the Regular Expression identity mapper. The `match-pattern` can be any regular expression supported by `javax.util.regex.Pattern`.

Configuring Proxied Authorization

Proxied authorization provides a standard control as defined in RFC 4370 (and an earlier Internet-Draft) for binding with the user credentials of a proxy, who carries out LDAP operations on behalf of other users. You might use proxied authorization, for example, to bind your application with its credentials, then carry out operations as the users who login to the application.

Proxied authorization is similar to the UNIX **sudo** command. The proxied operation is performed as if it were requested not by the user who did the bind, but by the proxied user. "Whether Proxy Authorization Allows an Operation on the Target" shows how this affects permissions.

Whether Proxy Authorization Allows an Operation on the Target

	Bind DN no access	**Bind DN has access**
Proxy ID no access	No	No

	Bind DN no access	Bind DN has access
Proxy ID has access	Yes	Yes

Note

When you configure resource limits as described in "*Setting Resource Limits*" in the *Administration Guide*, know that the resource limits do not change when the user proxies as another user. In other words, resource limits depend on the bind DN, not the proxy authorization identity.

Suppose you have an administrative directory client application that has an entry in the directory with DN `cn=My App,ou=Apps,dc=example,dc=com`. You can give that application the access rights and privileges to use proxied authorization. The default access control for the server lets authenticated users use the proxied authorization control.

Suppose also that when directory administrator, Kirsten Vaughan, logs in to your application to change Babs Jensen's entry, your application looks up Kirsten's entry, and finds that she has DN `uid=kvaughan,ou=People,dc=example,dc=com`. For the example commands in "To Configure Proxied Authorization", My App uses proxied authorization to make a change to Babs's entry as Kirsten.

To Configure Proxied Authorization

In order to carry out LDAP operations on behalf of another user, the user binding to a server needs:

- Permission to use the LDAP Proxy Authorization Control.

  Permissions are granted using access control instructions (ACIs). This calls for an ACI with a `targetcontrol` list that includes the Proxy Authorization Control OID `2.16.840.1.113730.3.4.18` that grants `allow(read)` permission to the user binding to the directory.

- Permission to proxy as the given authorization user.

  This calls for an ACI with a target scope that includes the entry of the authorization user that grants `allow(proxy)` permission to the user binding to the directory.

- The privilege to use proxied authorization.

  Privileges are granted using the `ds-privilege-name` attribute.

Follow these steps to configure proxied authorization for applications with DNs that match `cn=*,ou=Apps,dc=example,dc=com`:

1. If the global ACIs or global access policies do not allow access to use the Proxy Authorization Control, grant access to applications to use the Proxy Authorization control, and to use proxied authorization on behalf of other applications.

   The control has OID `2.16.840.1.113730.3.4.18`.

```
$ cat allow-proxy-authz.ldif
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetcontrol="2.16.840.1.113730.3.4.18")
  (version 3.0; acl "Apps can use the Proxy Authorization Control";
  allow(read) userdn="ldap:///cn=*,ou=Apps,dc=example,dc=com";)
aci: (target="ldap:///dc=example,dc=com") (targetattr ="*")
  (version 3.0; acl "Allow apps proxied auth";
  allow(proxy) (userdn = "ldap:///cn=*,ou=Apps,dc=example,dc=com");)

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 allow-proxy-authz.ldif
```

The latter ACI allows any user whose DN matches `cn=*,ou=Apps,dc=example,dc=com` to proxy as any user under the ACI target of `dc=example,dc=com`.

For example, `cn=My App,ou=Apps,dc=example,dc=com` can proxy as any user defined in the Example.com sample data, but cannot proxy as `cn=Directory Manager`. This is because all the users defined in the Example.com sample data have their accounts under `dc=example,dc=com`, and the target of the ACI includes `dc=example,dc=com`. `cn=Directory Manager` is defined in its own backend. The target of the ACI does not include `cn=Directory Manager`.

2. Grant the privilege to use proxied authorization to My App:

```
$ cat privilege-proxy-authz.ldif
dn: cn=My App,ou=Apps,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: proxied-auth

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 privilege-proxy-authz.ldif
```

3. Test that My App can use proxied authorization:

```
$ cat bjensen-proxy-authz-description.ldif
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Changed through proxied auth

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=My App,ou=Apps,dc=example,dc=com" \
 --bindPassword password \
 --proxyAs "dn:uid=kvaughan,ou=People,dc=example,dc=com" \
 bjensen-proxy-authz-description.ldif
# MODIFY operation successful for DN uid=bjensen,ou=People,dc=example,dc=com
```

If you need to map authorization identifiers with the `u:` form rather than `dn:`, you can set the identity mapper with the global configuration setting, proxied-authorization-identity-mapper. For example, if you get user ID values from the client, such as `bjensen`, you can configure a server to use the exact match identity mapper and match those to DNs based on an attribute of the entry. Use the **dsconfig** command interactively to determine the required settings.

Authenticating Client Applications With a Certificate

One alternative to simple binds with user name/password combinations consists of storing a digital certificate on the user entry, and using the certificate as credentials during the bind. You can use this mechanism, for example, to let applications bind without using passwords.

By setting up a secure connection with a certificate, the client is in effect authenticating to the server. The server must close the connection if it cannot trust the client certificate. However, in the process of establishing a secure connection, it does not identify the client to the DS server, because the secure connection is established by the JVM at the transport layer, independently of the LDAP protocol.

When binding with a certificate, the client must request the SASL External mechanism by which the DS server maps the certificate to the client entry in the directory. When it finds a match, the DS server sets the authorization identity for the connection to that of the client, and the bind is successful.

For the whole process of authenticating with a certificate to work smoothly, the DS server and the client must trust each others' certificates, the client certificate must be stored on the client entry in the directory, and the DS server must be configured to map the certificate to the client entry.

This section includes the following procedures and examples:

- "To Add Certificate Information to an Entry"
- "To Configure Certificate Mappers"

- "Authenticating With Client Certificates"

To Add Certificate Information to an Entry

Before you try to bind to DS servers using a certificate, create a certificate, and add the certificate attributes to the entry.

When you set up the directory server for evaluation, as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*, the server holds an entry, `cn=My App,ou=Apps,dc=example,dc=com`. Examples in this section use that entry, and use the Java **keytool** command to manage the certificate:

1. Create a certificate using the DN of the client entry as the `dname` string:

```
$ keytool \
 -genkeypair \
 -alias myapp-cert \
 -dname "cn=My App,ou=Apps,dc=example,dc=com" \
 -keystore /path/to/my-keystore \
 -storepass changeit \
 -keypass changeit
```

2. Get the certificate signed.

   If you cannot get the certificate signed by a CA, self-sign the certificate:

```
$ keytool \
 -selfcert \
 -alias myapp-cert \
 -validity 7300 \
 -keystore /path/to/my-keystore \
 -storepass changeit \
 -keypass changeit
```

3. Export the certificate to a file in binary format:

```
$ keytool \
 -export \
 -alias myapp-cert \
 -keystore /path/to/my-keystore \
 -storepass changeit \
 -keypass changeit \
 -file /path/to/myapp-cert.crt
Certificate stored in file </path/to/myapp-cert.crt>
```

4. Make note of the certificate MD5 fingerprint.

   Later in this procedure you update the client application entry with the MD5 fingerprint, referred to henceforth as `MD5_FINGERPRINT`:

```
$ keytool \
 -list \
 -v \
 -alias myapp-cert \
 -keystore /path/to/my-keystore \
 -storepass changeit | awk '/MD5/{print $2}'
MD5_FINGERPRINT
```

5. Modify the entry to add attributes related to the certificate.

   By default, you need the `userCertificate` value.

   If you want the server to map the certificate to its fingerprint, use the `ds-certificate-fingerprint` attribute. This example uses the MD5 fingerprint, which corresponds to the default setting for the fingerprint certificate mapper.

   To require that the certificate is issued by a known CA, use the `ds-certificate-issuer-dn` attribute. Use this to verify the certificate issuer whenever multiple CAs are trusted in order to prevent impersonation. Different CAs can issue certificates with the same subject DN, but not with the same issuer DN. In the example that follows, a self-signed certificate is used, so the issuer DN and the subject DN are the same. You must also specify the issuer attribute in the certificate mapper configuration, as shown below.

   If you want to map the certificate subject DN to an attribute of the entry, use the `ds-certificate-subject-dn` attribute:

```
$ cat addcert.ldif
dn: cn=My App,ou=Apps,dc=example,dc=com
changetype: modify
add: objectclass
objectclass: ds-certificate-user
-
add: ds-certificate-fingerprint
ds-certificate-fingerprint: MD5_FINGERPRINT
-
add: ds-certificate-issuer-dn
ds-certificate-issuer-dn: CN=My App, OU=Apps, DC=example, DC=com
-
add: ds-certificate-subject-dn
ds-certificate-subject-dn: CN=My App, OU=Apps, DC=example, DC=com
-
add: userCertificate;binary
userCertificate;binary:<file:///path/to/myapp-cert.crt

$ ldapmodify \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 addcert.ldif
```

6. Check your work:

```
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --baseDN dc=example,dc=com \
 "(cn=My App)"
dn: cn=My App,ou=Apps,dc=example,dc=com
ds-certificate-fingerprint: MD5_FINGERPRINT
userCertificate;binary:: ENCODED_CERT
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: ds-certificate-user
objectClass: top
ds-certificate-issuer-dn: CN=My App, OU=Apps, DC=example, DC=com
ds-certificate-subject-dn: CN=My App, OU=Apps, DC=example, DC=com
cn: My App
sn: App
```

7. When using a self-signed certificate, import the client certificate into the server truststore.

   When the client presents its certificate to the DS server, by default the server must trust the client certificate before it can accept the connection. If the DS server cannot trust the client certificate, it cannot establish a secure connection:

```
$ keytool \
 -import \
 -alias myapp-cert \
 -file /path/to/myapp-cert.crt \
 -keystore /path/to/opendj/config/keystore \
 -storetype PKCS12 \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -noprompt
Certificate was added to keystore
```

8. When using a certificate signed by a CA whose certificate is not delivered with the Java runtime environment, import the CA certificate into the Java runtime environment truststore, or into the server truststore as shown in the following example:

```
$ keytool \
 -import \
 -alias ca-cert \
 -file ca.crt \
 -keystore /path/to/opendj/config/keystore \
 -storetype PKCS12 \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -noprompt
Certificate was added to keystore
```

9. If you updated the truststore to add a certificate, restart the server to make sure that it reads the updated truststore and recognizes the certificate:

```
$ stop-ds --restart --quiet
```

To Configure Certificate Mappers

DS servers use certificate mappers during binds to establish a mapping between a client certificate and the entry that corresponds to that certificate. The certificate mappers shipped with the DS server include the following:

Fingerprint Certificate Mapper

Looks for the MD5 (default) or SHA-1 certificate fingerprint in an attribute of the entry (default: `ds-certificate-fingerprint`).

Subject Attribute To User Attribute Mapper

Looks for a match between an attribute of the certificate subject and an attribute of the entry (default: match `cn` in the certificate to `cn` on the entry, or match `emailAddress` in the certificate to `mail` on the entry).

Subject DN to User Attribute Certificate Mapper

Looks for the certificate subject DN in an attribute of the entry (default: `ds-certificate-subject-dn`).

Subject Equals DN Certificate Mapper

Looks for an entry whose DN matches the certificate subject DN.

If the default configurations for the certificate mappers are acceptable, you do not need to change them. They are enabled by default.

The following steps demonstrate how to change the Fingerprint Mapper default algorithm of MD5 to SHA1:

1. List the certificate mappers to retrieve the correct name:

```
$ dsconfig \
 list-certificate-mappers \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --trustAll \
 --no-prompt
Certificate Mapper                  : Type                                 : enabled
------------------------------------:--------------------------------------:--------
Fingerprint Mapper                  : fingerprint                          : true
Subject Attribute to User Attribute : subject-attribute-to-user-attribute : true
Subject DN to User Attribute        : subject-dn-to-user-attribute         : true
Subject Equals DN                   : subject-equals-dn                    : true
```

2. Examine the current configuration:

```
$ dsconfig \
 get-certificate-mapper-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --mapper-name "Fingerprint Mapper" \
 --trustAll \
 --no-prompt
Property              : Value(s)
----------------------:-----------------------------------------------------------
enabled               : true
fingerprint-algorithm : md5
fingerprint-attribute : ds-certificate-fingerprint
issuer-attribute      : The certificate issuer DN will not be verified.
user-base-dn          : The server performs the search in all public naming
                      : contexts.
```

3. Change the configuration as necessary.

   The following command updates the configuration to use SHA1, and to look for the issuer DN in the `ds-certificate-issuer-dn` attribute:

```
$ dsconfig \
 set-certificate-mapper-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --mapper-name "Fingerprint Mapper" \
 --set fingerprint-algorithm:sha1 \
 --set issuer-attribute:ds-certificate-issuer-dn \
 --trustAll \
 --no-prompt
```

4. Set the External SASL Mechanism Handler to use the appropriate certificate mapper (default: Subject Equals DN).

   Client applications use the SASL External mechanism during the bind to have the DS server set the authorization identifier based on the entry that matches the client certificate:

```
$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name External \
 --set certificate-mapper:"Fingerprint Mapper" \
 --trustAll \
 --no-prompt
```

Authenticating With Client Certificates

Instead of providing a bind DN and password as for simple authentication, use the SASL EXTERNAL authentication mechanism, and provide the certificate. As a test with example data, you can try an anonymous search, then try with certificate-based authentication.

Before you try this example, make sure the DS server is set up to accept StartTLS from clients, and that you have set up the client certificate as described above. Next, create a password PIN file for your client key store:

```
$ touch /path/to/my-keystore.pin
$ chmod 600 /path/to/my-keystore.pin
# Add the PIN (changeit) in cleartext on first and only line in the file:
$ vi /path/to/my-keystore.pin
```

Also, if the DS server uses a certificate for StartTLS that was not signed by a well-known CA, import the appropriate certificate into the client keystore, which can then double as a truststore. For example, if the DS server uses a self-signed certificate, import the server certificate into the keystore:

```
$ keytool \
 -export \
 -alias server-cert \
 -file server-cert.crt \
 -keystore /path/to/opendj/config/keystore \
 -storepass:file /path/to/opendj/config/keystore.pin \
 -storetype PKCS12
$ keytool \
 -import \
 -trustcacerts \
 -alias server-cert \
 -file server-cert.crt \
 -keystore /path/to/my-keystore \
 -storepass:file /path/to/my-keystore.pin \
 -noprompt
```

If the DS server uses a CA-signed certificate, but the CA is not well-known, import the CA certificate into your keystore:

```
$ keytool \
 -import \
 -trustcacerts \
 -alias ca-cert \
 -file ca-cert.crt \
 -keystore /path/to/my-keystore \
 -storepass:file /path/to/my-keystore.pin
```

Now that you can try the example, notice that the DS server does not return the `userPassword` value for an anonymous search:

```
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --baseDN dc=example,dc=com \
 --useStartTLS \
 --trustStorePath /path/to/my-keystore \
 --trustStorePasswordFile /path/to/my-keystore.pin \
 "(cn=My App)" \
 userPassword
dn: cn=My App,ou=Apps,dc=example,dc=com
```

DS servers do let users read the values of their own `userPassword` attributes after they bind successfully:

```
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --baseDN dc=example,dc=com \
 --useStartTLS \
 --saslOption mech="EXTERNAL" \
 --certNickName myapp-cert \
 --keyStorePath /path/to/my-keystore \
 --keyStorePasswordFile /path/to/my-keystore.pin \
 --trustStorePath /path/to/my-keystore \
 --trustStorePasswordFile /path/to/my-keystore.pin \
 "(cn=My App)" \
 userPassword
dn: cn=My App,ou=Apps,dc=example,dc=com
userPassword: {SSHA512}<hash>
```

You can also try the same test with other certificate mappers.

This example uses the fingerprint mapper:

```
$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name External \
 --set certificate-mapper:"Fingerprint Mapper" \
 --trustAll \
 --no-prompt
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --baseDN dc=example,dc=com \
 --useStartTLS \
 --saslOption mech="EXTERNAL" \
 --certNickName myapp-cert \
 --keyStorePath /path/to/my-keystore \
 --keyStorePasswordFile /path/to/my-keystore.pin \
 --trustStorePath /path/to/my-keystore \
 --trustStorePasswordFile /path/to/my-keystore.pin \
 "(cn=My App)" \
 userPassword
dn: cn=My App,ou=Apps,dc=example,dc=com
userPassword: {SSHA512}<hash>
```

This example uses the subject attribute to user attribute mapper:

```
$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name External \
 --set certificate-mapper:"Subject Attribute to User Attribute" \
 --trustAll \
 --no-prompt
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --baseDN dc=example,dc=com \
 --useStartTLS \
 --saslOption mech="EXTERNAL" \
 --certNickName myapp-cert \
 --keyStorePath /path/to/my-keystore \
 --keyStorePasswordFile /path/to/my-keystore.pin \
 --trustStorePath /path/to/my-keystore \
 --trustStorePasswordFile /path/to/my-keystore.pin \
 "(cn=My App)" \
 userPassword
dn: cn=My App,ou=Apps,dc=example,dc=com
userPassword: {SSHA512}<hash>
```

This example uses the subject DN to user attribute mapper:

```
$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name External \
 --set certificate-mapper:"Subject DN to User Attribute" \
 --trustAll \
 --no-prompt
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --baseDN dc=example,dc=com \
 --useStartTLS \
 --saslOption mech="EXTERNAL" \
 --certNickName myapp-cert \
 --keyStorePath /path/to/my-keystore \
 --keyStorePasswordFile /path/to/my-keystore.pin \
 --trustStorePath /path/to/my-keystore \
 --trustStorePasswordFile /path/to/my-keystore.pin \
 "(cn=My App)" \
 userPassword
dn: cn=My App,ou=Apps,dc=example,dc=com
userPassword: {SSHA512}<hash>
```

Chapter 5

Using LDAP Schema

LDAP services are based on X.500 Directory Services, which are telecommunications standards. In telecommunications, interoperability is paramount. Competitors must cooperate to the extent that they use each others' systems. For directory services, the protocols for exchanging data and the descriptions of the data are standardized. LDAP defines *schema* that describe both what attributes a given LDAP entry must have and may optionally have, and also what attribute values can contain and how they can be matched. Formal schema definitions protect interoperability when many applications read and write to the same directory service. Directory data are much easier to share as long as you understand how to use LDAP schema.

"*Managing Schema*" in the *Administration Guide* covers LDAP schema from the server administrator's perspective. Administrators can update LDAP directory schema. DS servers support a large number of standard schema definitions by default. Administrators can also adjust how strictly each DS server applies schema definitions.

This chapter covers LDAP schema from the script developer's perspective. As a script developer, you use the available schema and accept the server's application of schema when updating directory entries.

In this chapter you will learn how to:

- Look up available schemas
- Understand what the schemas allow
- Understand and resolve errors that arise due to schema violations

Getting Schema Information

Directory servers publish information about services they provide as operational attributes of the *root DSE*. The root DSE is the entry with an empty string DN, `""`. DSE is an acronym for DSA-Specific Entry. DSA is an acronym for Directory System Agent. The DSE differs by server, but is generally nearly identical for replicas.

DS servers publish the DN of the entry holding schema definitions as the value of the attribute `subschemaSubentry`. This is shown in "Finding the Schema Entry".

Finding the Schema Entry

Look up the schema DN:

```
$ ldapsearch --port 1389 --baseDN "" --searchScope base "(&)" subschemaSubentry
dn:
subschemaSubentry: cn=schema
```

By default, the DN for the schema entry is `cn=schema`.

The schema entry has the following attributes whose values are schema definitions:

`attributeTypes`

Attribute type definitions describe attributes of directory entries, such as `givenName` or `mail`.

`objectClasses`

Object class definitions identify the attribute types that an entry must have, and may have. Examples of object classes include `person` and `organizationalUnit`. Object classes inherit from other object classes. For example, `inetOrgPerson` inherits from `person`.

Object classes are specified as values of an entry's `objectClass` attribute.

An object class can be one of the following:

- *Structural* object classes define the core structure of the entry, generally representing a real-world object.

  By default, DS directory entries have a single structural object class or at least a single line of structural object class inheritance.

  The `person` object class is structural, for example.

- *Auxiliary* object classes define additional characteristics of entries.

  The `posixAccount` object class is auxiliary, for example.

- *Abstract* object classes define base characteristics for other object classes to inherit, and cannot themselves inherit from other object classes.

  The `top` object class from which others inherit is abstract, for example.

`ldapSyntaxes`

An *attribute syntax* constrains what directory clients can store as attribute values.

`matchingRules`

A `Matching rule` determines how the directory server compares attribute values to assertion values for LDAP search and LDAP compare operations.

For example, in a search having the filter `(uid=bjensen)` the assertion value is `bjensen`.

`nameForms`

A *name form* specifies which attribute can be used as the relative DN (RDN) for a structural object class.

`dITStructureRules`

A *DIT structure rule* defines a relationship between directory entries by identifying the name form allowed for subordinate entries of a given superior entry.

Reading an Object Class Schema Definition

The schema entry in a server is large because it contains all of the schema definitions. Filter the results when reading a specific schema definition.

The example below reads the definition for the `person` object class:

```
$ grep \'person\' <(ldapsearch \
 --port 1389 \
 --baseDN "cn=schema" \
 --searchScope base \
 "(&)" \
 objectClasses)
objectClasses: ( 2.5.6.6 NAME 'person' SUP top STRUCTURAL MUST ( sn $ cn ) MAY ( userPassword $
 telephoneNumber $ seeAlso $ description ) X-ORIGIN 'RFC 4519' X-SCHEMA-FILE '00-core.ldif' )
```

Notice the use of the object class name in **grep \'person\'** to filter search results.

The object class defines which attributes an entry of that object class *must* have and which attributes the entry *may* optionally have. A `person` entry must have a `cn` and an `sn` attribute. A `person` entry may optionally have `userPassword`, `telephoneNumber`, `seeAlso`, and `description` attributes.

To determine definitions of those attributes, read the LDAP schema as demonstrated in "Reading Schema Definitions for an Attribute".

Reading Schema Definitions for an Attribute

The following example shows you how to read the schema definition for the `cn` attribute:

```
$ grep \'cn\' <(ldapsearch \
 --port 1389 \
 --baseDN "cn=schema" \
 --searchScope base \
 "(&)" \
 attributeTypes)
attributeTypes: ( 2.5.4.3 NAME ( 'cn' 'commonName' ) SUP name X-ORIGIN 'RFC 4519' X-SCHEMA-FILE '00-
core.ldif' )
```

The `cn` attribute inherits its definition from the `name` attribute. That attribute definition indicates attribute syntax and matching rules as shown in the following example:

```
$ grep \'name\' <(ldapsearch \
 --port 1389 \
 --baseDN "cn=schema" \
 --searchScope base \
 "(&)" \
 attributeTypes)
attributeTypes: ( 2.5.4.41 NAME 'name' EQUALITY caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch SYNTAX
 1.3.6.1.4.1.1466.115.121.1.15 X-ORIGIN 'RFC 4519' X-SCHEMA-FILE '00-core.ldif' )
```

This means that the server ignores case when matching a common name value. Use the OID to read the syntax as shown in the following example:

```
$ grep 1.3.6.1.4.1.1466.115.121.1.15 <(ldapsearch \
 --port 1389 \
 --baseDN "cn=schema" \
 --searchScope base \
 "(&)" \
 ldapSyntaxes)
ldapSyntaxes: ( 1.3.6.1.4.1.1466.115.121.1.15 DESC 'Directory String' X-ORIGIN 'RFC 4517' )
```

Taken together with the information for the `name` attribute, the common name attribute value is a Directory String of at most 32,768 characters. For details about syntaxes, read RFC 4517, *Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules*. That document describes a Directory String as one or more UTF-8 characters.

Respecting LDAP Schema

For the sake of interoperability and to avoid polluting directory data, scripts and applications should respect LDAP schema. In the simplest case, scripts and applications can use the schemas already defined.

DS servers do accept updates to schema definitions over LDAP while the server is running. This means that when a new application calls for attributes that are not yet defined by existing directory schemas, the directory administrator can easily add them as described in "Updating Directory Schema" in the *Administration Guide* as long as the new definitions do not conflict with existing definitions.

General purpose applications handle many different types of data. Such applications must manage schema compliance at run time. Software development kits provide mechanisms for reading schema definitions at run time and checking whether entry data is valid according to the schema definitions.

Many scripts do not require run time schema checking. In such cases it is enough properly to handle schema-related LDAP result codes when writing to the directory:

LDAP result code: 17 (Undefined attribute type)

The requested operation failed because it referenced an attribute that is not defined in the server schema.

LDAP result code: 18 (Inappropriate matching)

The requested operation failed because it attempted to perform an inappropriate type of matching against an attribute.

LDAP result code: 20 (Attribute or value exists)

The requested operation failed because it would have resulted in a conflict with an existing attribute or attribute value in the target entry.

For example, the request tried to add a second value to a single-valued attribute.

LDAP result code: 21 (Invalid attribute syntax)

The requested operation failed because it violated the syntax for a specified attribute.

LDAP result code: 34 (Invalid DN syntax)

The requested operation failed because it would have resulted in an entry with an invalid or malformed DN.

LDAP result code: 64 (Naming violation)

The requested operation failed because it would have violated the server's naming configuration.

For example, the request did not respect a name form definition.

LDAP result code: 65 (Object class violation)

The requested operation failed because it would have resulted in an entry that violated the server schema.

For example, the request tried to remove a required attribute, or tried to add an attribute that is not allowed.

LDAP result code: 69 (Object class mods prohibited)

The requested operation failed because it would have modified] the object classes associated with an entry in an illegal manner.

When you encounter an error, take the time to read the additional information. The additional information from a server often suffices to allow you to resolve the problem directly.

"Object Class Violations" and "Invalid Attribute Syntax" show some common problems that can result from schema violations.

Object Class Violations

A number of schema violations show up as object class violations. The following request fails to add an `undefined` attribute:

```
$ cat undefined.ldif
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: undefined
undefined: This attribute is not defined.

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 undefined.ldif
# The LDAP modify request failed: 65 (Object Class Violation)
# Additional Information:  Entry uid=bjensen,ou=People,dc=example,dc=com cannot be modified because the
 resulting entry would have violated the server schema: Entry "uid=bjensen,ou=People,dc=example,dc=com"
 violates the schema because it contains attribute "undefined" which is not allowed by any of the object
 classes in the entry
```

The solution in this case is to make sure that the `undefined` attribute is defined and that it is allowed by one of the object classes defined for the entry.

The following request fails to add a second structural object class:

```
$ cat second-structural.ldif
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: organizationalUnit

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 second-structural.ldif
# The LDAP modify request failed: 65 (Object Class Violation)
# Additional Information:  Entry uid=bjensen,ou=People,dc=example,dc=com cannot be modified because the
 resulting entry would have violated the server schema: Entry "uid=bjensen,ou=People,dc=example,dc=com"
 violates the schema because it contains multiple conflicting structural object classes "inetOrgPerson"
 and "organizationalUnit". Only a single structural object class is allowed in an entry
```

The solution in this case is to define only one structural object class for the entry. Either Babs Jensen is a person or an organizational unit, but not both.

Invalid Attribute Syntax

The following request fails to add an empty string as a common name attribute value:

```
$ cat empty-cn.ldif
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: cn
cn:

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 empty-cn.ldif
# The LDAP modify request failed: 21 (Invalid Attribute Syntax)
# Additional Information:  When attempting to modify entry uid=bjensen,ou=People,dc=example,dc=com to add
 one or more values for attribute cn, value "" was found to be invalid according to the associated syntax:
 The operation attempted to assign a zero-length value to an attribute with the directory string syntax
```

As mentioned in "Reading Schema Definitions for an Attribute", a Directory String has one or more UTF-8 characters.

Abusing LDAP Schema

Follow the suggestions in "Respecting LDAP Schema" as much as possible. In particular follow these rules of thumb:

- Test with a private DS server to resolve schema issues before going live.
- Adapt your scripts and applications to avoid violating schema definitions.
- When existing schemas are not sufficient, request schema updates to add definitions that do not conflict with any already in use.

When it is not possible to respect the schema definitions, you can sometimes work around LDAP schema constraints without changing the server configuration. The schema defines an `extensibleObject` object class. The `extensibleObject` object class is auxiliary. It effectively allows entries to hold any user attribute, even attributes that are not defined in the schema.

Working Around Restrictions With ExtensibleObject

The following example adds one attribute that is undefined and another that is not allowed:

```
$ cat extensible-object.ldif
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: extensibleObject
-
add: undefined
undefined: This attribute is not defined in the LDAP schema.
-
add: serialNumber
serialNumber: This attribute is not allowed according to the object classes.

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 extensible-object.ldif
# MODIFY operation successful for DN uid=bjensen,ou=People,dc=example,dc=com
```

Use of the `extensibleObject` object class opens the door to abuse and can prevent interoperability. Restrict its use to cases where no better alternative is available.

Standard Schema Included With DS Servers

DS servers provide many standard schema definitions. For documentation on the available schema definitions, see LDAP Schema Reference. The LDAP schema elements are defined in these LDIF files in the `/path/to/opendj/db/schema/` directory:

`00-core.ldif`

This file contains a core set of attribute type and object class definitions from the following Internet-Drafts, RFCs, and standards:

draft-boreham-numsubordinates
draft-findlay-ldap-groupofentries
draft-good-ldap-changelog
draft-howard-namedobject
draft-ietf-ldup-subentry
draft-wahl-ldap-adminaddr
RFC 1274
RFC 2079
RFC 2256
RFC 2798
RFC 3045
RFC 3296
RFC 3671
RFC 3672
RFC 4512

RFC 4519
RFC 4523
RFC 4524
RFC 4530
RFC 5020
X.501

`01-pwpolicy.ldif`

This file contains schema definitions from draft-behera-ldap-password-policy (Draft 09), which defines a mechanism for storing password policy information in an LDAP directory server.

`02-config.ldif`

This file contains the attribute type and objectclass definitions for use with the server configuration.

`03-changelog.ldif`

This file contains schema definitions from draft-good-ldap-changelog, which defines a mechanism for storing information about changes to directory server data.

`03-rfc2713.ldif`

This file contains schema definitions from RFC 2713, which defines a mechanism for storing serialized Java objects in the directory server.

`03-rfc2714.ldif`

This file contains schema definitions from RFC 2714, which defines a mechanism for storing CORBA objects in the directory server.

`03-rfc2739.ldif`

This file contains schema definitions from RFC 2739, which defines a mechanism for storing calendar and vCard objects in the directory server. Note that the definition in RFC 2739 contains a number of errors, and this schema file has been altered from the standard definition in order to fix a number of those problems.

`03-rfc2926.ldif`

This file contains schema definitions from RFC 2926, which defines a mechanism for mapping between Service Location Protocol (SLP) advertisements and LDAP.

`03-rfc3112.ldif`

This file contains schema definitions from RFC 3112, which defines the authentication password schema.

`03-rfc3712.ldif`

This file contains schema definitions from RFC 3712, which defines a mechanism for storing printer information in the directory server.

`03-uddiv3.ldif`

This file contains schema definitions from RFC 4403, which defines a mechanism for storing UDDIv3 information in the directory server.

`04-rfc2307bis.ldif`

This file contains schema definitions from draft-howard-rfc2307bis, which defines a mechanism for storing naming service information in the directory server.

`05-rfc4876.ldif`

This file contains schema definitions from RFC 4876, which defines a schema for storing Directory User Agent (DUA) profiles and preferences in the directory server.

`05-samba.ldif`

This file contains schema definitions required when storing Samba user accounts in the directory server.

`05-solaris.ldif`

This file contains schema definitions required for Solaris and OpenSolaris LDAP naming services.

`06-compat.ldif`

This file contains the attribute type and objectclass definitions for use with the server configuration.

Chapter 6

Working With Groups of Entries

DS servers support several methods of grouping entries in the directory. Static groups list their members, whereas dynamic groups look up their membership based on an LDAP filter. DS servers supports virtual static groups, which use a dynamic group-style definition, but allow applications to list group members as if the group were static.

When listing entries in static groups, you must also have a mechanism for removing entries from the list when they are deleted or modified in ways that end their membership. DS servers make that possible through their *referential integrity* capabilities.

In this chapter you will learn how to:

- Create static (enumerated) groups
- Create dynamic groups based on LDAP URLs
- Create virtual static groups that make dynamic groups look like static groups
- Look up group membership efficiently
- Work with nested groups
- Make sure that when an entry is deleted or modified, the server updates affected groups appropriately

Tip

The examples in this chapter are written with the assumption that an `ou=Groups,dc=example,dc=com` entry already exists. If you set up the directory server for evaluation as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*, then you already have the entry. If you generated data during setup and did not create an organizational unit for groups yet, create the entry before you try the examples:

```
$ cat groups.ldif
dn: ou=Groups,dc=example,dc=com
objectClass: organizationalunit
objectClass: top
ou: Groups

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 groups.ldif
```

Creating Static Groups

A *static group* is expressed as an entry that enumerates all the entries that belong to the group. Static group entries grow as their membership increases.

Tip

Large static groups can be a performance bottleneck. The recommended way to avoid the issue is to use dynamic groups instead as described in "Creating Dynamic Groups". If using dynamic groups is not an option for a deployment with large static groups that are updated regularly, use an entry cache. For details, see "Caching Large, Frequently Used Entries" in the *Administration Guide*.

Static group entries can take the standard object class `groupOfNames` where each `member` attribute value is a distinguished name of an entry, or `groupOfUniqueNames` where each `uniqueMember` attribute value has Name and Optional UID syntax. (Name and Optional UID syntax values are a DN optionally followed by `#BitString`. The *BitString*, such as `'0101111101'B`, serves to distinguish the entry from another entry having the same DN, which can occur when the original entry was deleted and a new entry created with the same DN.) Like other LDAP attributes, `member` and `uniqueMember` attributes take sets of unique values.

Static group entries can also have the object class `groupOfEntries`, which is like `groupOfNames` except that it is designed to allow groups not to have members.

When creating a group entry, use `groupOfNames` or `groupOfEntries` where possible.

To create a static group, add a group entry such as the following to the directory:

```
$ cat static.ldif
dn: cn=My Static Group,ou=Groups,dc=example,dc=com
cn: My Static Group
objectClass: groupOfNames
objectClass: top
ou: Groups
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 static.ldif
```

To change group membership, modify the values of the membership attribute:

```
$ cat add2group.ldif
dn: cn=My Static Group,ou=Groups,dc=example,dc=com
changetype: modify
add: member
member: uid=scarter,ou=People,dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 add2group.ldif
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(cn=My Static Group)"
dn: cn=My Static Group,ou=Groups,dc=example,dc=com
objectClass: top
objectClass: groupOfNames
cn: My Static Group
ou: Groups
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
member: uid=scarter,ou=People,dc=example,dc=com
```

RFC 4519 says a `groupOfNames` entry must have at least one member. Although DS servers allows you to create a `groupOfNames` without members, strictly speaking, that behavior is not standard. Alternatively, you can use the `groupOfEntries` object class as shown in the following example:

```
$ cat group-of-entries.ldif
dn: cn=Initially Empty Static Group,ou=Groups,dc=example,dc=com
cn: Initially Empty Static Group
objectClass: groupOfEntries
objectClass: top
ou: Groups

$ ldapmodify \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 group-of-entries.ldif
$ cat add-members.ldif
dn: cn=Initially Empty Static Group,ou=Groups,dc=example,dc=com
changetype: modify
add: member
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
member: uid=scarter,ou=People,dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 add-members.ldif
```

Creating Dynamic Groups

A *dynamic group* specifies members using LDAP URLs. Dynamic groups entries can stay small even as their membership increases.

Dynamic group entries take the `groupOfURLs` object class, with one or more `memberURL` values specifying LDAP URLs to identify group members.

To create a dynamic group, add a group entry such as the following to the directory.

The following example builds a dynamic group of entries, effectively matching the filter `"(l=San Francisco)"` (users whose location is San Francisco). Change the filter if your data is different, and so no entries have `l: San Francisco`:

```
$ cat dynamic.ldif
dn: cn=My Dynamic Group,ou=Groups,dc=example,dc=com
cn: My Dynamic Group
objectClass: top
objectClass: groupOfURLs
ou: Groups
memberURL: ldap:///ou=People,dc=example,dc=com??sub?l=San Francisco

$ ldapmodify \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 dynamic.ldif
```

Group membership changes dynamically as entries change to match the `memberURL` values:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com \
 "(&(uid=*jensen)(isMemberOf=cn=My Dynamic Group,ou=Groups,dc=example,dc=com))" 1.1
dn: uid=bjensen,ou=People,dc=example,dc=com

dn: uid=rjensen,ou=People,dc=example,dc=com
$ cat move-ajensen.ldif
dn: uid=ajensen,ou=People,dc=example,dc=com
changetype: modify
replace: l
l: San Francisco

$ ldapmodify \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 move-ajensen.ldif
$ ldapsearch --port 1389 --baseDN dc=example,dc=com \
 "(&(uid=*jensen)(isMemberOf=cn=My Dynamic Group,ou=Groups,dc=example,dc=com))" 1.1
dn: uid=ajensen,ou=People,dc=example,dc=com

dn: uid=bjensen,ou=People,dc=example,dc=com

dn: uid=rjensen,ou=People,dc=example,dc=com
```

Creating Virtual Static Groups

DS servers let you create *virtual static groups* Virtual static groups allow applications to see dynamic groups as what appear to be static groups.

The virtual static group takes auxiliary object class `ds-virtual-static-group`. Virtual static groups also take either the object class `groupOfNames`, or `groupOfUniqueNames`, but instead of having `member` or `uniqueMember` attributes, have `ds-target-group-dn` attributes pointing to other groups.

Generating the list of members can be resource-intensive for large groups, so by default, you cannot retrieve the list of members. You can change this with the **dsconfig** command by setting the `Virtual Static member` or `Virtual Static uniqueMember` property:

```
$ dsconfig \
 set-virtual-attribute-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --name "Virtual Static member" \
 --set allow-retrieving-membership:true \
 --trustAll \
 --no-prompt
```

The following example creates a virtual static group, and reads the group entry with all members:

```
$ cat virtual.ldif
dn: cn=Virtual Static,ou=Groups,dc=example,dc=com
cn: Virtual Static
objectclass: top
objectclass: groupOfNames
objectclass: ds-virtual-static-group
ds-target-group-dn: cn=My Dynamic Group,ou=Groups,dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 virtual.ldif
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(cn=Virtual Static)"
dn: cn=Virtual Static,ou=Groups,dc=example,dc=com
objectClass: top
objectClass: groupOfNames
objectClass: ds-virtual-static-group
cn: Virtual Static
ds-target-group-dn: cn=My Dynamic Group,ou=Groups,dc=example,dc=com
member: uid=abergin,ou=People,dc=example,dc=com
member: uid=ajensen,ou=People,dc=example,dc=com
member: uid=aknutson,ou=People,dc=example,dc=com
member: uid=awalker,ou=People,dc=example,dc=com
member: uid=aworrell,ou=People,dc=example,dc=com
member: uid=bplante,ou=People,dc=example,dc=com
member: uid=btalbot,ou=People,dc=example,dc=com
member: uid=cwallace,ou=People,dc=example,dc=com
member: uid=dakers,ou=People,dc=example,dc=com
member: uid=dthorud,ou=People,dc=example,dc=com
member: uid=ewalker,ou=People,dc=example,dc=com
member: uid=gfarmer,ou=People,dc=example,dc=com
member: uid=jbourke,ou=People,dc=example,dc=com
member: uid=jcampaig,ou=People,dc=example,dc=com
member: uid=jmuffly,ou=People,dc=example,dc=com
member: uid=jreuter,ou=People,dc=example,dc=com
member: uid=jwalker,ou=People,dc=example,dc=com
member: uid=kcarter,ou=People,dc=example,dc=com
```

```
member: uid=kschmith,ou=People,dc=example,dc=com
member: uid=mjablons,ou=People,dc=example,dc=com
member: uid=mlangdon,ou=People,dc=example,dc=com
member: uid=mschneid,ou=People,dc=example,dc=com
member: uid=mtalbot,ou=People,dc=example,dc=com
member: uid=mtyler,ou=People,dc=example,dc=com
member: uid=mwhite,ou=People,dc=example,dc=com
member: uid=pshelton,ou=People,dc=example,dc=com
member: uid=rjensen,ou=People,dc=example,dc=com
member: uid=tlabonte,ou=People,dc=example,dc=com
member: uid=tschmith,ou=People,dc=example,dc=com
```

Looking Up Group Membership

DS servers let you look up which groups a user belongs to with the `isMemberOf` attribute:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=bjensen)" isMemberOf
dn: uid=bjensen,ou=People,dc=example,dc=com
isMemberOf: cn=My Static Group,ou=Groups,dc=example,dc=com
isMemberOf: cn=Carpoolers,ou=Self Service,ou=Groups,dc=example,dc=com
```

You must request `isMemberOf` explicitly.

Nesting Groups Within Groups

DS servers let you nest groups. The following example shows a group of groups of managers and administrators:

```
$ cat the-big-shots.ldif
dn: cn=The Big Shots,ou=Groups,dc=example,dc=com
cn: The Big Shots
objectClass: groupOfNames
objectClass: top
ou: Groups
member: cn=Accounting Managers,ou=groups,dc=example,dc=com
member: cn=Directory Administrators,ou=Groups,dc=example,dc=com
member: cn=HR Managers,ou=groups,dc=example,dc=com
member: cn=PD Managers,ou=groups,dc=example,dc=com
member: cn=QA Managers,ou=groups,dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 the-big-shots.ldif
```

Although not shown in the example above, DS servers let you nest groups within nested groups, too.

DS servers let you create dynamic groups of groups. The following example shows a group of other groups. The members of this group are themselves groups, not users:

```
$ cat group-of-groups.ldif
dn: cn=Group of Groups,ou=Groups,dc=example,dc=com
cn: Group of Groups
objectClass: top
objectClass: groupOfURLs
ou: Groups
memberURL: ldap:///ou=Groups,dc=example,dc=com??sub?ou=Groups

$ ldapmodify \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 group-of-groups.ldif
```

Use the `isMemberOf` attribute to determine what groups a member belongs to, as described in "Looking Up Group Membership". The following example requests groups that Kirsten Vaughan belongs to:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=kvaughan)" isMemberOf
dn: uid=kvaughan,ou=People,dc=example,dc=com
isMemberOf: cn=Directory Administrators,ou=Groups,dc=example,dc=com
isMemberOf: cn=HR Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com
```

Notice that Kirsten is a member of the group of groups of managers and administrators.

Notice also that Kirsten does not belong to the group of groups. The members of that group are groups, not users. The following example requests the groups that the directory administrators group belongs to:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(cn=Directory Administrators)" isMemberOf
dn: cn=Directory Administrators,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com
```

The following example shows which groups each group belong to:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(ou=Groups)" isMemberOf
dn: ou=Groups,dc=example,dc=com

dn: cn=Accounting Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com

dn: cn=Directory Administrators,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com
```

```
dn: cn=HR Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com

dn: cn=PD Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com

dn: cn=QA Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com

dn: cn=My Static Group,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com

dn: cn=My Dynamic Group,ou=Groups,dc=example,dc=com

dn: cn=The Big Shots,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com

dn: cn=Group of Groups,ou=Groups,dc=example,dc=com
```

Notice that the group of groups is not a member of itself.

Configuring Referential Integrity

When you delete or rename an entry that belongs to static groups, that entry's DN must be removed or changed in each group it belongs to. You can configure the server to resolve membership on your behalf after the change operation succeeds by enabling referential integrity.

Referential integrity functionality is implemented as a plugin. The referential integrity plugin is disabled by default. To enable the plugin, use the **dsconfig** command:

```
$ dsconfig \
 set-plugin-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --plugin-name "Referential Integrity" \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

With the plugin enabled, referential integrity resolves group membership automatically:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(cn=My Static Group)"
dn: cn=My Static Group,ou=Groups,dc=example,dc=com
objectClass: top
objectClass: groupOfNames
cn: My Static Group
ou: Groups
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
member: uid=scarter,ou=People,dc=example,dc=com
$ ldapdelete \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 uid=scarter,ou=People,dc=example,dc=com
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(cn=My Static Group)"
dn: cn=My Static Group,ou=Groups,dc=example,dc=com
ou: Groups
objectClass: groupOfNames
objectClass: top
cn: My Static Group
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
```

By default, the referential integrity plugin is configured to manage `member` and `uniqueMember` attributes. These attributes take values that are DNs, and are indexed for equality by default for the default backend. Before you add an additional attribute to manage, make sure that it has DN syntax and that it is indexed for equality. DS servers require that the attribute be indexed because an unindexed search for integrity would potentially consume too many of the server's resources. Attribute syntax is explained in "*Managing Schema*" in the *Administration Guide*. For instructions on indexing attributes, see "Configuring and Rebuilding Indexes" in the *Administration Guide*.

You can also configure the referential integrity plugin to check that new entries added to groups actually exist in the directory by setting the `check-references` property to `true`. You can specify additional criteria once you have activated the check. To ensure that entries added must match a filter, set the `check-references-filter-criteria` to identify the attribute and the filter. For example, you can specify that group members must be person entries by setting `check-references-filter-criteria` to `member:(objectclass=person)`. To ensure that entries must be located in the same naming context, set `check-references-scope-criteria` to `naming-context`.

Chapter 7

Working With Virtual and Collective Attributes

DS servers support virtual attributes with dynamically generated values. Some virtual attributes are defined by default. You can also define your own. DS servers also support standard collective attributes, described in RFC 3671, allowing entries to share common, read-only attribute values.

In this chapter you will learn how to define virtual and collective attributes.

Virtual Attributes

Virtual attributes augment directory entries with attribute values that the DS server computes or obtains dynamically. Virtual attribute values do not exist in persistent storage. They help to limit the amount of data that needs to be stored and are great for some uses, such as determining the groups a users belongs to or adding an ETag to an entry.

Do not index virtual attributes. Virtual attribute values generated by the server when they are read. They are not designed to be stored in a persistent index.

Since you do not index virtual attributes, searching on a virtual attribute can result in an unindexed search. For an unindexed search, the DS directory server potentially has to go through all entries to look for candidate matches. Looking through all entries is resource-intensive for large directories. By default, a directory server allows only the directory superuser to perform unindexed searches. Generally avoid searches that use a simple filter with a virtual attribute. Instead, consider the alternatives. You can assign a password policy to a group as described in "To Assign a Password Policy to a Group" in the *Administration Guide*. The procedure uses a virtual attribute only in a subtree specification filter. If you must use a virtual attribute in a search filter, use it in a complex search filter after narrowing the search by filtering on an indexed attribute. For example, the following filter first narrows the search based on the user's ID before checking group membership. Make sure that the user performing the search has access to read `isMemberOf` in the results:

```
(&(uid=user-id)(isMemberOf=group-dn))
```

Two virtual attributes, `entryDN` and `isMemberOf`, can also be used in simple equality filters. The following example shows how to add access to read `isMemberOf` and then run a search that returns the common names for members of a group:

```
$ cat admins-isMemberOf.ldif
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr="isMemberOf")(version 3.0; acl "See isMemberOf"; allow (read,search,compare)
  groupdn= "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com";)

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 admins-isMemberOf.ldif
$ ldapsearch \
 --port 1389 \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN dc=example,dc=com \
 "(isMemberOf=cn=Directory Administrators,ou=Groups,dc=example,dc=com)" \
 cn
dn: uid=hmiller,ou=People,dc=example,dc=com
cn: Harry Miller

dn: uid=kvaughan,ou=People,dc=example,dc=com
cn: Kirsten Vaughan

dn: uid=rdaugherty,ou=People,dc=example,dc=com
cn: Robert Daugherty
```

DS servers define the following virtual attributes by default:

`entryDN`

The value is the DN of the entry.

`entryUUID`

Provides a universally unique identifier for the entry.

`etag`

Entity tag as defined in RFC 2616, useful for checking whether an entry has changed since you last read it from the directory.

`hasSubordinates`

Boolean. Indicates whether the entry has children.

`numSubordinates`

Provides the number of direct child entries.

`isMemberOf`

Identifies groups the entry belongs to.

By default, the server generates `isMemberOf` on user entries (entries that have the object class `person`), and on group entries (entries that have the object class `groupOfNames`, `groupOfUniqueNames`, or `groupOfEntries`). You can change this by editing the filter property of the `isMemberOf` virtual attribute configuration.

`member`

Generated for virtual static groups.

`uniqueMember`

Generated for virtual static groups.

`pwdPolicySubentry`

Identifies the password policy that applies to the entry.

By default, a directory server assigns the directory superuser the password policy with DN `cn=Root Password Policy,cn=Password Policies,cn=config`, and regular users the password policy with DN `cn=Default Password Policy,cn=Password Policies,cn=config`. See "*Configuring Password Policy*" in the *Administration Guide* for information on configuring and assigning password policies.

The default global access control instructions prevent this operational attribute from being visible to normal users.

`subschemaSubentry`

References the schema definitions.

`collectiveAttributeSubentries`

References applicable collective attribute definitions.

`governingStructureRule`

References the rule on what type of subordinates the entry can have.

`structuralObjectClass`

References the structural object class for the entry.

These virtual attributes are typically operational, so you get them back from a search only when you request them:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(dc=example)"
dn: dc=example,dc=com
dc: example
objectClass: domain
objectClass: top
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(dc=example)" numSubordinates
dn: dc=example,dc=com
numSubordinates: 11
```

You can use the existing virtual attribute types to create your own virtual attributes, and you can also use the `user-defined` type to create your own virtual attribute types. The virtual attribute is defined by the server configuration, which is not replicated:

```
$ dsconfig \
 create-virtual-attribute \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --name "Served By Description" \
 --type user-defined \
 --set enabled:true \
 --set attribute-type:description \
 --set base-dn:dc=example,dc=com \
 --set value:"Served by opendj.example.com" \
 --trustAll \
 --no-prompt
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=wlutz)" description
dn: uid=wlutz,ou=People,dc=example,dc=com
description: Served by opendj.example.com
```

Collective attributes cover many use cases better than virtual attributes.

Collective Attributes

Collective attributes provide a standard mechanism for defining attributes that appear on all the entries in a subtree potentially filtered by object class. Standard collective attribute type names have the prefix `c-`.

DS servers extend collective attributes to make them easier to use. You can define any DS attribute as collective with the `;collective` attribute option. You can use LDAP filters in your subtree specification for fine-grained control over entries that have the collective attributes.

You can have entries inherit attributes from other entries through collective attributes. You establish the relationship between entries either by indicating the attribute holding the DN of the entry from which to inherit the attributes, or by specifying how to construct the RDN of the entry from which to inherit the attributes.

"To Add Privileges for a Group of Administrators" in the *Administration Guide* demonstrates setting administrative privileges with collective attributes. The following examples demonstrate additional ways to use collective attributes:

- "Class of Service With Collective Attributes"
- "Inheriting an Attribute From the Manager's Entry"
- "Inheriting Attributes From the Locality"

Class of Service With Collective Attributes

This example defines attributes that specify services available to a user depending on their service level.

Note

The following example depends on the `cos` object class, and the `classOfService` attribute type.

This example shows collective attributes that depend on the `classOfService` attribute values:

- For entries with `classOfService: bronze`, `mailQuota` is set to 1 GB, and `diskQuota` is set to 10 GB.
- For entries with `classOfService: silver`, `mailQuota` is set to 5 GB, and `diskQuota` is set to 50 GB.
- For entries with `classOfService: gold`, `mailQuota` is set to 10 GB, and `diskQuota` is set to 100 GB.

You define collective attributes in the user data using a subentry. In other words, collective attributes can be replicated. Collective attributes use attributes defined in the directory schema. The following LDIF shows the schema definitions:

```
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( example-class-of-service-attribute-type
  NAME 'classOfService'
  EQUALITY caseIgnoreMatch
  ORDERING caseIgnoreOrderingMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE
  USAGE userApplications
  X-ORIGIN 'DS Documentation Examples' )
-
add: attributeTypes
attributeTypes: ( example-class-of-service-disk-quota
  NAME 'diskQuota'
  EQUALITY caseIgnoreMatch
  ORDERING caseIgnoreOrderingMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  USAGE userApplications
  X-ORIGIN 'DS Documentation Examples' )
-
add: attributeTypes
attributeTypes: ( example-class-of-service-mail-quota
  NAME 'mailQuota'
  EQUALITY caseIgnoreMatch
  ORDERING caseIgnoreOrderingMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  USAGE userApplications
```

```
  X-ORIGIN 'DS Documentation Examples' )
-
add: objectClasses
objectClasses: ( example-class-of-service-object-class
  NAME 'cos'
  SUP top
  AUXILIARY
  MAY ( classOfService $ diskQuota $ mailQuota )
  X-ORIGIN 'DS Documentation Examples' )
```

The collective attribute definitions set the quotas depending on class of service:

```
dn: cn=Bronze Class of Service,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Bronze Class of Service
diskQuota;collective: 10 GB
mailQuota;collective: 1 GB
subtreeSpecification: { base "ou=People", specificationFilter "(classOfService=bronze)" }

dn: cn=Silver Class of Service,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Silver Class of Service
diskQuota;collective: 50 GB
mailQuota;collective: 5 GB
subtreeSpecification: { base "ou=People", specificationFilter "(classOfService=silver)" }

dn: cn=Gold Class of Service,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Gold Class of Service
diskQuota;collective: 100 GB
mailQuota;collective: 10 GB
subtreeSpecification: { base "ou=People", specificationFilter "(classOfService=gold)" }
```

With the collective attributes defined, you can see the results on user entries. To try this example for yourself, set up a directory server for evaluation as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=bjensen)" \
 classOfService mailQuota diskQuota
dn: uid=bjensen,ou=People,dc=example,dc=com
classOfService: bronze
mailQuota: 1 GB
diskQuota: 10 GB
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=kvaughan)" \
 classOfService mailQuota diskQuota
dn: uid=kvaughan,ou=People,dc=example,dc=com
classOfService: silver
mailQuota: 5 GB
diskQuota: 50 GB
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=scarter)" \
 classOfService mailQuota diskQuota
dn: uid=scarter,ou=People,dc=example,dc=com
classOfService: gold
mailQuota: 10 GB
diskQuota: 100 GB
```

For details on how subentries apply, see "Understanding Subentry Scope".

Inheriting an Attribute From the Manager's Entry

This example demonstrates how to instruct the server to set an employee's department number using the manager's department number.

For this example, the relationship between employee entries and manager entries is based on the manager attributes on employee entries. Each `manager` attribute on an employee's entry specifies the DN of the manager's entry. The server retrieves the department number from the manager's entry to populate the attribute on the employee's entry.

The collective attribute subentry that specifies the relationship looks like this:

```
dn: cn=Inherit Department Number From Manager,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: inheritedCollectiveAttributeSubentry
objectClass: inheritedFromDNCollectiveAttributeSubentry
cn: Inherit Department Number From Manager
subtreeSpecification: { base "ou=People" }
inheritFromDNAttribute: manager
inheritAttribute: departmentNumber
```

This entry specifies that users inherit department number from their manager.

Babs Jensen's manager is Torrey Rigden:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
manager: uid=trigden, ou=People, dc=example,dc=com
```

Torrey's department number is 3001:

```
dn: uid=trigden,ou=People,dc=example,dc=com
departmentNumber: 3001
```

Babs inherits her department number from Torrey. To try this example for yourself, set up a directory server for evaluation as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=bjensen)" departmentNumber
dn: uid=bjensen,ou=People,dc=example,dc=com
departmentNumber: 3001
```

For details on how subentries apply, see "Understanding Subentry Scope".

Inheriting Attributes From the Locality

This example demonstrates how to instruct a server to set a user's language preferences and street address based on locality.

For this example, the relationship between entries is based on locality. The collective attribute subentry specifies how to construct the RDN of the object holding the attribute values to inherit:

```
dn: cn=Inherit From Locality,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: inheritedCollectiveAttributeSubentry
objectClass: inheritedFromRDNCollectiveAttributeSubentry
cn: Inherit From Locality
subtreeSpecification: { base "ou=People" }
inheritFromBaseRDN: ou=Locations
inheritFromRDNAttribute: l
inheritFromRDNType: l
inheritAttribute: preferredLanguage
inheritAttribute: street
collectiveConflictBehavior: real-overrides-virtual
```

This specifies that the RDN of the entry to inherit attributes from is like `l=localityName,ou=Locations`, where *localityName* is the value of the `l` (`localityName`) attribute on the user's entry.

In other words, if the user's entry has `l: Bristol`, then the RDN of the entry from which to inherit attributes starts with `l=Bristol,ou=Locations`. The actual entry looks like this:

```
dn: l=Bristol,ou=Locations,dc=example,dc=com
objectClass: top
objectClass: locality
objectClass: extensibleObject
l: Bristol
street: Broad Quay House, Prince Street
preferredLanguage: en-gb
```

The subentry specifies two attributes to inherit for preferred language and street address.

The object class `extensibleObject` allows the entry to take a preferred language. (The object class `extensibleObject` means, "Let me add whatever attributes I want." It is usually better practice to add

your own auxiliary object class if you need to decorate an entry with more attributes. The shortcut is taken here as the focus of this example is not schema extension, but instead how to use collective attributes.)

Notice the last line of the collective attribute subentry:

```
collectiveConflictBehavior: real-overrides-virtual
```

This line indicates that if a collective attribute clashes with a real attribute, the real value takes precedence over the virtual, collective value. You can set `collectiveConflictBehavior` to `virtual-overrides-real` for the opposite precedence, or to `merge-real-and-virtual` to keep both sets of values.

In this example, users can set their own language preferences. When users set language preferences manually, the collective attribute subentry is configured to give the user's settings precedence over the locality-based setting, which is only a default guess.

Sam Carter is located in Bristol. Sam has specified no preferred languages:

```
dn: uid=scarter,ou=People,dc=example,dc=com
l: Bristol
```

Sam inherits the street address and preferred language from the Bristol locality. To try this example for yourself, set up a directory server for evaluation as shown in "To Set Up a Directory Server for Evaluation" in the *Installation Guide*:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=scarter)" preferredLanguage street
dn: uid=scarter,ou=People,dc=example,dc=com
preferredLanguage: en-gb
street: Broad Quay House, Prince Street
```

Babs's locality is San Francisco. Babs prefers English, but also knows Korean:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
preferredLanguage: en, ko;q=0.8
l: San Francisco
```

Babs inherits the street address from the San Francisco locality, but keeps her language preferences:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=bjensen)" preferredLanguage street
dn: uid=bjensen,ou=People,dc=example,dc=com
preferredLanguage: en, ko;q=0.8
street: 201 Mission Street Suite 2900
```

For details on how subentries apply, see "Understanding Subentry Scope".

Understanding Subentry Scope

LDAP subentries reside with the user data and so are replicated. Subentries hold operational data. They are not visible in search results unless explicitly requested. This section describes how a subentry's `subtreeSpecification` attribute defines the scope of the subtree that the subentry applies to.

An LDAP subentry's subtree specification identifies a subset of entries in a branch of the DIT. The subentry scope is these entries. In other words, these are the entries that the subentry affects.

The attribute value for a `subtreeSpecification` optionally includes the following parameters:

`base`

Indicates the entry, *relative to the subentry's parent*, at the base of the subtree.

By default, the base is the subentry's parent.

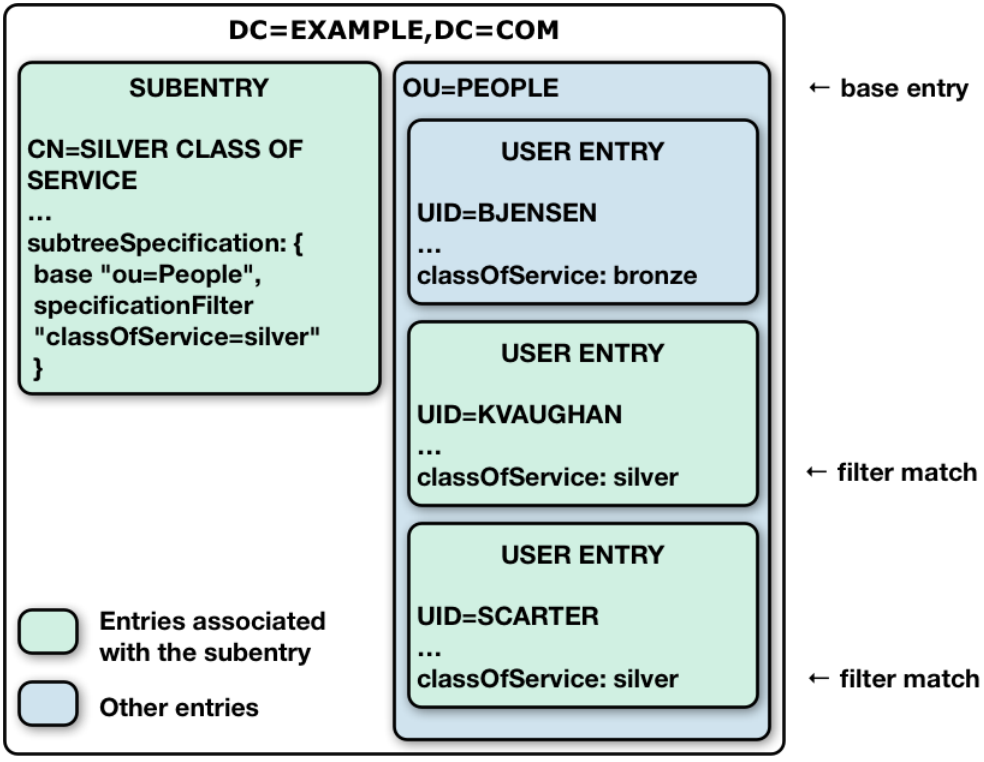
`specificationFilter`

Indicates an LDAP filter. Entries matching the filter are in scope.

By default, all entries under the base entry are in scope.

"Subtree Associated With a Subentry" illustrates these characteristics for an example collective attribute subentry.

Subtree Associated With a Subentry



Notice that the base of `ou=People` on the subentry `cn=Silver Class of Service,dc=example,dc=com` indicates that the base entry is `ou=People,dc=example,dc=com`.

The filter `"(classOfService=silver)"` means that Kirsten Vaughan and Sam Carter's entries are in scope. Babs Jensen's entry, with `classOfService: bronze` does not match and is therefore not in scope. The `ou=People` organizational unit entry does not have a `classOfService` attribute, and so is not in scope, either.

Chapter 8

Working With Referrals

Referrals point directory clients to another directory container, which can be another directory server running elsewhere, or another container on the same server. The client receiving a referral must use the other container to complete the request.

Note

Some clients follow referrals on your behalf by default. The DS commands do not follow referrals.

Referrals are used, for example, when directory data is temporarily unavailable due to maintenance. Referrals can also be used when a container holds only some of the directory data for a suffix and points to other containers for branches whose data is not available locally.

In this chapter you will learn how to:

- Allow administrators to manage referrals
- Add referrals
- Remove referrals

About Referrals

Referrals are implemented as entries with LDAP URL `ref` attribute values that point elsewhere. The `ref` attribute type is required by the `referral` object class. The `referral` object class is structural, however, and therefore cannot by default be added to an entry that already has a structural object class defined. When adding a `ref` attribute to an entry that already has a structural object class, use the `extensibleObject` auxiliary object class.

When a referral is set, DS servers return the referral to requests that target the affected entry or its child entries. Client applications must be capable of following the referral returned. When the directory server responds with referrals to LDAP URLs, the client can construct new operations and try them again.

The Manage DSAIT control makes it possible to access the referral entry, rather than get a referral when accessing the entry. It has OID `2.16.840.1.113730.3.4.2`. The control is described in RFC 3296.

Managing Referrals

Suppose the entries below `ou=Subscribers,dc=example,dc=com` are stored on a different directory server at `referral.example.com:1389`. You can create a LDAP referral to reference the remote entries.

Before creating the LDAP referral, give directory administrators access to use the Manage DSAIT control and to manage the `ref` attribute:

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --add global-aci:"(targetcontrol=\"2.16.840.1.113730.3.4.2\")\
 (version 3.0; acl \"Allow Manage DSAIT control\"; allow(read)\
 groupdn=\"ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com\";)" \
 --trustAll \
 --no-prompt
$ cat allow-ref-access.ldif
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (target="ldap:///dc=example,dc=com")(targetattr="ref")
  (version 3.0; acl "Admins can manage referrals"; allow(all)
  (groupdn = "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com");)

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 allow-ref-access.ldif
```

To create an LDAP referral, either create a referral entry, or add the `extensibleObject` object class and the `ref` attribute with an LDAP URL to an existing entry. The example above creates a referral entry at `ou=Subscribers,dc=example,dc=com`:

```
$ cat referral.ldif
dn: ou=Subscribers,dc=example,dc=com
objectClass: top
objectClass: extensibleObject
objectClass: organizationalUnit
ou: Subscribers
ref: ldap://referral.example.com:1389/ou=Subscribers,dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 referral.ldif
```

DS servers can now return a referral for operations under `ou=Subscribers`:

```
$ ldapsearch --port 1389 --baseDN ou=subscribers,dc=example,dc=com "(uid=*)"
# The LDAP search request failed: 10 (Referral)
# Additional Information:  A referral entry ou=Subscribers,dc=example,dc=com indicates that the operation
 must be processed at a different server
# Matched DN:  ou=Subscribers,dc=example,dc=com
```

To access the entry instead of the referral, use the Manage DSAIT control as a user with access to request the control:

```
$ ldapsearch \
 --control 2.16.840.1.113730.3.4.2:true \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN ou=subscribers,dc=example,dc=com \
 "(&)" \
 ref
dn: ou=Subscribers,dc=example,dc=com
ref: ldap://referral.example.com:1389/ou=Subscribers,dc=example,dc=com
```

You can use the Manage DSAIT control to change the referral with the **ldapmodify** command:

```
$ cat change-referral.ldif
dn: ou=Subscribers,dc=example,dc=com
changetype: modify
replace: ref
ref: ldap://opendj.example.com:1389/ou=People,dc=example,dc=com

$ ldapmodify \
 --control 2.16.840.1.113730.3.4.2:true \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 change-referral.ldif
```

Chapter 9

Writing a Server Plugin

DS servers have many features that are implemented as server *plugins*. A server plugin is a library that can be plugged in to an installed server and immediately configured for use.

In this chapter you will learn:

- Enough about the DS plugin architecture to begin writing plugins
- How to build and use the example plugin delivered with the server
- How the parts of the example plugin project fit together

Important

ForgeRock supports customers using standard plugins delivered as part of DS server software.

If you deploy with custom plugins and need support in production, contact info@forgerock.com in advance to determine how your deployment can be supported.

About DS Server Plugins

DS server plugins are Java libraries compiled against the DS server Java API. Plugins are built to be configured as part of the server and to be invoked at specific points in the lifecycle of a client request, or in the server process lifecycle.

Note

The DS server Java API has interface stability: Evolving, as described in "ForgeRock Product Stability Labels" in the *Release Notes*.

This means that a server plugin built with one version of DS server software will not necessarily work or even compile with a different version of the server.

Plugin Types

Plugin types correspond to the points where the server invokes the plugin.

For the full list of plugin invocation points, see the Javadoc for `PluginType`. The following list summarizes the plugin invocation points:

- At server startup and shutdown
- Before and after data export and import
- Immediately after a client connection is established or is closed
- Before processing begins on an LDAP operation (to change an incoming request before it is decoded)
- Before core processing for LDAP operations (to change the way the server handles the operation)
- After core processing for LDAP operations (where the plugin can access all information about the operation including the impact it has on the targeted entry)
- When a subordinate entry is deleted as part of a subtree delete or moved or renamed as part of a modify DN operation
- Before sending intermediate and search responses
- After sending a result

A plugin's types are specified in its configuration, and can therefore be modified at runtime.

Plugin Configuration

Server plugin configuration is managed with the same configuration framework that is used for DS server configuration.

The DS configuration framework has these characteristics:

- LDAP schemas govern what attributes can be used in plugin configuration entries.

  For all configuration attributes that are specific to a plugin, the plugin should have its own object class and attributes defined in the server LDAP schema. Having configuration entries governed by schemas makes it possible for the server to identify and prevent configuration errors.

  For plugins, having schema for configuration attributes means that an important part of plugin installation is making the schema definitions available to the DS server.

- The plugin configuration is declared in XML files.

  The XML specifies configuration properties and their documentation, and also inheritance relationships.

  The XML Schema Definition files (.xsd files) for the namespaces used in these documents are part of the DS Maven Plugin. They are published as part of the source code of that module, not in the locations corresponding to their namespace identifiers.

  In other words, you can find `admin.xsd`, for example, in the DS source code. Its XML namespace identifier (`http://opendj.forgerock.org/admin`) is not a URL that you can browse to.

For details, see also "Configuration".

- Compilation generates the server-side and client-side APIs to access the plugin configuration from the XML.

  To use the server-side APIs in a plugin project, first generate and compile them, and include the classes on the project classpath. You can see how the `opendj-maven-plugin` is used to generate sources from the XML in the example plugin project sources. The process is described in "Maven Project".

  When a plugin is loaded in the DS server, the client-side APIs are available to configuration tools like the **dsconfig** command. Directory administrators can configure a custom plugin in the same way they configure other server components.

- The framework supports internationalization.

A complete plugin project, such as the example plugin, therefore includes LDAP schema definitions, XML configuration definitions, Java plugin code, and Java resource bundles.

Trying the Example Server Plugin

The example plugin is bundled with DS servers as `example-plugin.zip`, which holds a Maven-based project. The example plugin is a startup plugin that displays a "Hello World" message when the directory server starts. For general information about DS server plugins, read "About DS Server Plugins". For more specific information, read "About the Example Plugin Project Files".

To Try the Example Plugin

Follow these steps to try the example plugin:

1. Install the DS server as described in Installation Guide.
2. Install Apache Maven 3.0.5 or later.

   When you finish, make sure **mvn** is on your PATH:

   ```
   $ mvn -version
   Apache Maven version
   Maven home: /path/to/maven
   Java version: ...
   ```

3. Unpack the example plugin project sources:

   ```
   $ unzip /path/to/opendj/example-plugin.zip
   Archive:  /path/to/opendj/example-plugin.zip
      creating: opendj-server-example-plugin/
   ...
   ```

4. Build the example plugin:

```
$ cd opendj-server-example-plugin/
$ mvn install
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
...
```

5. Install the example plugin in the DS server:

```
$ cd /path/to/opendj

# Stop the server before installing the example plugin:
$ bin/stop-ds

# Unpack the plugin files into the proper locations of the server layout,
# skipping the base directory.
# The following example works with bsdtar,
# which might require installing a bsdtar package.
$ bsdtar -xvf \
 /path/to/opendj-server-example-plugin/target/opendj-server-example-plugin-6.5.6.zip \
 -s'|[^/]*/||'
x README.example.plugin
x config/
x config/example-plugin.ldif
x db/
x db/schema/
x db/schema/99-example-plugin.ldif
x lib/
x lib/extensions/
x lib/extensions/opendj-server-example-plugin-6.5.6.jar
x lib/extensions/...

# Start the server and create the plugin configuration:
$ bin/start-ds
$ bin/dsconfig \
 create-plugin \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --plugin-name "Example Plugin" \
 --type example \
 --set enabled:true \
 --set plugin-type:startup \
 --trustAll \
 --no-prompt
...
INFO: Loaded extension from file
 '/path/to/opendj/lib/extensions/opendj-server-example-plugin-6.5.6.jar'
 (build <unknown>, revision <unknown>)
```

Notice the locations where the example plugin files are unpacked. The locations must follow the server conventions in order for the DS server to recognize the plugin.

For the example plugin, you see that:

- Schema definitions are unpacked into the `db/schema/` directory.
- Plugin .jar files and the .jar files they depend on are unpacked into `lib/extensions/`.

Also notice that after the plugin configuration is created the DS server has loaded the plugin as an extension.

6. Restart the DS server to see the startup message from the plugin:

```
$ bin/stop-ds --restart
...
... msg=Example plugin message 'HELLO WORLD'.
...
```

7. Now that you have seen the example plugin display its message, see "About the Example Plugin Project Files" to understand the key parts of the example plugin project.

About the Example Plugin Project Files

The example plugin project builds a server plugin that displays a "Hello World" message when the DS server starts, as shown in "Trying the Example Server Plugin". This section describes the example plugin project. For general information about DS server plugins, read "About DS Server Plugins" instead.

Maven Project

The DS example server plugin is an Apache Maven project.

As you can see in the `pom.xml` file for the project, the plugin depends on the DS server module.

The plugin project uses these ForgeRock Maven plugins:

- The `i18n-maven-plugin` generates message source files from properties files in the resource bundle.

  This plugin must run in order to resolve static imports from `com.example.opendj.ExamplePluginMessages`.

- The `opendj-maven-plugin` generates source files, manifest files, and resource bundles from the configuration declarations in the XML configuration files.

  This plugin must run in order to resolve imports from `com.example.opendj.server.ExamplePluginCfg`.

Configuration

The example plugin has the following configuration files:

`src/main/assembly/descriptor.xml`

This defines how to bundle the different components of the plugin in a layout appropriate for installation into DS servers.

`src/main/assembly/config/example-plugin.ldif`

This shows an example configuration entry for the plugin.

`src/main/assembly/db/schema/99-example-plugin.ldif`

This defines all object classes and attribute types that are specific to the example plugin configuration. The XML file that defines the configuration also specifies how configuration properties map to the object class and attribute type defined here for the LDAP representation of the configuration, using the definitions from this addition to the LDAP schema.

If your plugin has no configuration attributes of its own, then there is no need to extend the LDAP schema.

For more information on defining your own LDAP schemas, see "*Managing Schema*" in the *Administration Guide*.

`src/main/java/com/example/opendj/ExamplePluginConfiguration.xml`

This defines the configuration interface to the example plugin, and an LDAP profile that maps the plugin configuration to an LDAP entry.

Notice that the name ends in `Configuration.xml`, which is the expected suffix for configuration files.

The configuration definition has these characteristics:

- The attributes of the `<managed-object>` element define XML namespaces, a (singular) name and plural name for the plugin, and the Java-related inheritance of the implementation to generate. A *managed object* is a configurable component of DS servers.

  A managed object definition covers the object's structure and inheritance, and is like a class in Java. The actual managed object is like an instance of an object in Java. Its configuration maps to a single LDAP entry in the configuration backend `cn=config`.

  Notice that the `<profile>` element defines how the whole object maps to an LDAP entry in the configuration. The `<profile>` element is mandatory, and should include an LDAP profile.

  The `name` and `plural-name` properties are used to identify the managed object definition. They are also used when generating Java class names. Names must be a lowercase sequence of words separated by hyphens.

  The `package` property specifies the Java package name for generated code.

The `extends` property identifies a parent definition that the current definition inherits.

- The mandatory `<synopsis>` element provides a brief description of the managed object.

  If a longer description is required, add a `<description>`, which can include XHTML markup. The `<description>` is used in addition to the synopsis, so there is no need to duplicate the synopsis in the description.

- The `<property>` element defines a property specific to this example plugin, including its purpose, its the default value, its type, and how the property maps to an LDAP attribute in the configuration entry.

  The `name` attribute is used to identify the property in the configuration.

- The `<property-override>` element sets the pre-defined property `java-class` to a specific value, namely that of the fully qualified implementation class.

The XML-based configuration files are more powerful than this short explanation suggests. See the documentation in the XML schema definitions for more details about the elements and attributes.

When the example plugin project is built, generated Java properties files are written in `target/generated-resources/`, and generated Java source files are written in `target/generated-sources/`.

`src/main/java/com/example/opendj/Package.xml`

This defines the package-level short description used in generated `package-info.java` source files.

Implementation Code

The plugin implementation is found in `src/main/java/com/example/opendj/ExamplePlugin.java`. It relies on the DS server Java API.

Note

The DS server Java API has interface stability: Evolving, as described in "ForgeRock Product Stability Labels" in the *Release Notes*.

This means that a server plugin built with one version of DS server software will not necessarily work or even compile with a different version of the server.

`ExamplePlugin` statically imports everything from the generated message implementation sources. Resolution of `ExamplePluginMessages.*` fails until the implementation is generated by the `i18n-maven-plugin`.

`ExamplePlugin` extends `DirectoryServerPlugin` with its own type of configuration, `ExamplePluginCfg`. The implementation for `ExamplePluginCfg` is generated from the configuration declared in XML. Therefore, resolution of `ExamplePluginCfg` fails until the sources are generated by the `opendj-maven-plugin`.

`ExamplePlugin` implements `ConfigurationChangeListener` so the plugin can be notified of changes to its configuration. The plugin can then potentially update its configuration without the need to restart the plugin or the DS server.

The example plugin stores a reference to its configuration in the private `config` object. Your plugins should follow this example.

When the server first configures the plugin, it does so by calling the `initializePlugin` method. This method must do the following things:

- Perform checks that the configuration framework cannot do for the plugin, such as checking dependencies between properties or checking system state (whether some file is writable, or if there is sufficient disk space, for example).

  The example plugin checks that its type is `startup`.

- Initialize the plugin, if necessary.

  The example plugin has nothing to initialize.

- Register to receive configuration change notifications by using the `addExampleChangeListener()` method.

- Cache the current state of the configuration.

  The example plugin assigns the configuration to its private `config` object.

On subsequent configuration changes, the server calls the `isConfigurationChangeAcceptable()` method. If the method returns true because the configuration is valid, the server calls `applyConfigurationChange()` method.

Although the example plugin's `isConfigurationChangeAcceptable()` method always returns true, other plugins might need to perform checks that the framework cannot, in the same way they perform checks during initialization.

In the `applyConfigurationChange()` method the plugin must modify its configuration as necessary. The example plugin can handle configuration changes without further intervention by the administrator. Other plugins might require administrative intervention because changes can be made that can only be taken into account at plugin initialization.

In the example plugin, the method that extends the server's behavior is the `doStartup()` method. Which method is implemented depends on what class the plugin extends. For example, a password validator extending `PasswordValidator` would implement a `passwordIsAcceptable()` method.

Internationalization

In the example plugin, localized messages are found in the resource bundle under `src/main/resources/com/example/opendj/`.

The `LocalizedLogger` in the plugin implementation is capable of selecting the right messages from the resource bundle based on the locale for the server.

If the server runs in a French locale, then the plugin can log messages in French when a translation exists. Otherwise, it falls back to English messages, as those are the messages defined for the default locale.

Chapter 10

Embedding the Server

You can embed DS servers in your Java applications, and manage them with the embedded server APIs. An embedded server runs in the same JVM and memory space as your application, and can be accessed directly through Java APIs.

In this chapter you will examine sample Java code to learn how to:

- Set up an embedded server using the default configuration
- Start and stop an embedded server
- Begin to configure an embedded server using APIs rather than files

Important

The server APIs used in this chapter are of Evolving interface stability, as described in "ForgeRock Product Stability Labels" in the *Release Notes*.

Be prepared to accommodate incompatible changes in minor releases.

Before Trying the Embedded Server Samples

Before you try the embedded server examples, prepare your development environment:

- Install a development environment (IDE) with Java, Apache Maven, and Git support.

  Common IDEs include Eclipse and IntelliJ IDEA.
- If necessary, register for access to the appropriate version of the DS project source code.
- If necessary, install Apache Maven 3.0.5 or later command-line tools.
- If necessary, install **git** command-line tools.

Obtaining the Sample Code

The embedded server sample code is part of the DS project code. After you have prepared your development environment as described in "Before Trying the Embedded Server Samples", obtain the project code that contains the samples.

Get the DS project code for your version of the server. The following example uses the **git** command to clone the latest public version of the project code:

```
$ git clone ssh://git@stash.forgerock.org:7999/opendj/opendj.git
Cloning into 'opendj'...
```

Build the DS project code to ensure that the necessary generated code for the configuration APIs is available. The following example uses the **mvn** command to build the project code:

```
$ cd opendj
$ mvn install
...
[INFO] BUILD SUCCESS
```

You can find the sample code in the `opendj-embedded-server-examples` module.

Setting Up an Embedded Server

After completing the steps described in "Obtaining the Sample Code", try the server setup sample, listed in "Set Up an Embedded Server".

To Set Up an Embedded Server

1. Configure the IDE to run the `SetupServer.main()` method.

   There is no need to have the IDE rebuild the class or its dependencies, as you already built those as described in "Obtaining the Sample Code".

   The main method takes the following arguments in this order:

   `OPENDJ_ARCHIVE`

   The full path on your system to the server archive, found in the file `opendj/opendj-server/target/opendj-6.5.6.zip`.

   `SERVER_ROOT_DIR`

   The full path to the directory where the sample class should install the server, such as `/path/to/opendj/opendj-embedded-server-examples/target/opendj`.

   `LDAP_PORT`

   Port number for the LDAP connection handler.

   Default: 1500.

   To use the default value, pass an empty string, `""`, as the value of this argument.

`ADMIN_PORT`

Port number for the LDAP connection handler.

Default: 4500.

To use the default value, pass an empty string, `""`, as the value of this argument.

`SETUP_MODE`

Set this to one of the following values:

- `with_sample_backend`: adds a `userRoot` backend with base DN `dc=example,dc=com`, and imports 10 generated user entries.
- `with_profiles`: applies the `am-cts`, `am-identity-store`, and `idm-repo` profiles described in "*Using Directory Server Setup Profiles*" in the *Installation Guide*.

2. (Optional) Set breakpoints if you want to step through the code.
3. Run the sample.

   The sample configures the server in the location you specified.

   If successful, the sample displays the following output:

```
Validating parameters..... Done
Configuring certificates..... Done
Configuring server..... Done
Configuring profile AM 6.5 CTS data store..... Done
Configuring profile AM 6.5 identity data store..... Done
Configuring profile IDM 6.5 external repository..... Done
Setup completed successfully.
```

If you prefer to run the samples directly without an IDE, execute the file `opendj-embedded-server-examples/README`.

Set Up an Embedded Server

```
/*
 * Copyright 2016-2018 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */

package org.forgerock.opendj.examples;

import static org.forgerock.opendj.server.embedded.ConfigParameters.configParams;
import static org.forgerock.opendj.server.embedded.ConnectionParameters.connectionParams;
import static org.forgerock.opendj.server.embedded.EmbeddedDirectoryServer.manageEmbeddedDirectoryServer;
import static org.forgerock.opendj.setup.model.DirectoryServerSetup.importSampleData;
```

```
import static org.forgerock.opendj.setup.model.DirectoryServerSetup.localBackend;
import static org.forgerock.opendj.setup.model.Profile.newSetupProfile;
import static org.forgerock.opendj.setup.model.Profile.parameters;

import java.io.File;
import java.nio.file.Path;
import java.nio.file.Paths;

import org.forgerock.i18n.LocalizableMessage;
import org.forgerock.opendj.server.embedded.EmbeddedDirectoryServer;
import org.forgerock.opendj.server.embedded.EmbeddedDirectoryServerException;
import org.forgerock.opendj.setup.model.Profile;
import org.forgerock.opendj.setup.model.SetupConsole;

/** Setup a server from a OpenDJ archive using the EmbeddedDirectoryServer class. */
public final class SetupServer {

    /**
     * Main method which can be used to setup an embedded OpenDJ server.
     * <p>
     * Provided arguments must respect the following synopsis:
     * {@literal OPENDJ_ARCHIVE SERVER_ROOT_DIR LDAP_PORT ADMIN_PORT SETUP_MODE} where:
     * <ul>
     *     <li>OPENDJ_ARCHIVE represents the path to the zip archive that is resulting from a maven build
     *     <li>SERVER_ROOT_DIR represents the directory where OpenDJ will be installed. Because the
 archive contains
     *     the "opendj" directory, it is mandatory to provide a server root directory that is named
 "opendj"
     *     (the archive will be automatically extracted in the parent directory of the provided server
 root directory)
     *     <li>LDAP_PORT represents the LDAP connection handler port of the server to be setup. If empty
 or
     *     not valid, {@literal 1500} will be used</li>
     *     <li>ADMIN_PORT represents the administration connector port of the server to be setup. If empty
 or
     *     not valid, {@literal 4500} will be used
     *     <li>SETUP_MODE represents how the directory server should be configured, must be set to either:
     *     <ul>
     *         <li>{@literal with_sample_backend}
     *         <li>{@literal with_profiles}
     *     </ul>
     * </ul>
     *
     * Setup a server with a sample backend:
     * <pre>
     *     /path/to/opendj-archive.zip /path/to/opendj 1389 4444 with_sample_backend
     * </pre>
     *
     * Setup a server with profiles (using port default values):
     * <pre>
     *     /path/to/opendj-archive.zip /path/to/opendj "" "" with_profiles
     * </pre>
     *
     * @param args
     *            The command line arguments
     * @throws Exception
     *          If an error occurs
     */
    public static void main(final String[] args) throws Exception {
```

```
        if (args.length != 5) {
            printUsageAndExit();
        }

        int i = 0;
        final String openDJArchive = args[i++];
        final Path serverRootDir = Paths.get(args[i++]);
        final int ldapPort = parseIntOrDefault(args[i++], 1500);
        final int adminPort = parseIntOrDefault(args[i++], 4500);

        switch (args[i]) {
        case "with_profiles":
            setupServerWithProfiles(openDJArchive, serverRootDir, ldapPort, adminPort);
            break;
        case "with_sample_backend":
            setupServerWithSampleBackend(openDJArchive, serverRootDir, ldapPort, adminPort);
            break;
        default:
            printUsageAndExit();
        }
    }

    private static void setupServerWithProfiles(
            String openDJArchive, Path serverRootDir, int ldapPort, int adminPort) throws Exception {
        final EmbeddedDirectoryServer embeddedDs =
                createEmbeddedDirectoryServer(openDJArchive, serverRootDir, ldapPort, adminPort);
        final Path profilesPath = serverRootDir.resolve("template").resolve("setup-profiles");

        // Declare setup profiles
        final Profile amCts = newSetupProfile(
                "am-cts",
                profilesPath.resolve("AM").resolve("cts").resolve("6.5"));
        final Profile amIdentityStore = newSetupProfile(
                "am-identity-store",
                profilesPath.resolve("AM").resolve("identity-store").resolve("6.5"));
        final Profile idmRepoProfile = newSetupProfile(
                "idm-repo",
                profilesPath.resolve("IDM").resolve("repo").resolve("6.5"));

        // Load profile parameters (by running parameters.groovy script) and provide their values
        amCts.resolveParameterValues(
            serverRootDir,
            parameters().parameter("amCtsAdminPassword", "thisIsSecret!")
                        .parameter("useAmReaper", false)
                        .parameter("ttlAttribute", "coreTokenTtlDate"));
        amIdentityStore.resolveParameterValues(
            serverRootDir,
            parameters().parameter("amIdentityStoreAdminPassword", "thisIsSecret!")
                        .parameter("domain", "my.company.com"));
        idmRepoProfile.resolveParameterValues(
            serverRootDir,
            parameters().parameter("domain", "my.company.com"));

        System.out.println();
        // Setup the server with profiles
        embeddedDs.initializeConfiguration()
                  .console(sampleConsole())
                  .withProfiles(amCts,
                                amIdentityStore,
```

```
                                idmRepoProfile)
                    .setup();
        System.out.println("Setup completed successfully.");
    }

    private static void setupServerWithSampleBackend(
            String openDJArchive, Path serverRootDir, int ldapPort, int adminPort) throws Exception {
        System.out.println();
        createEmbeddedDirectoryServer(openDJArchive, serverRootDir, ldapPort, adminPort)
            .initializeConfiguration()
            .console(sampleConsole())

.addLocalBackend(localBackend("userRoot").importData(importSampleData(10).addBaseDns("dc=example,dc=com")))
            .setup();
        System.out.println("Setup completed successfully.");
    }

    private static EmbeddedDirectoryServer createEmbeddedDirectoryServer(
            String openDJArchive, Path serverRootDir, int ldapPort, int adminPort)
            throws EmbeddedDirectoryServerException {
        return manageEmbeddedDirectoryServer(
                configParams()
                        .serverRootDirectory(serverRootDir)
                        .configurationFile(serverRootDir.resolve("config").resolve("config.ldif")),
                connectionParams()
                        .hostName("localhost")
                        .ldapPort(ldapPort)
                        .bindDn("cn=Directory Manager")
                        .bindPassword("password")
                        .adminPort(adminPort),
                System.out,
                System.err)
        .extractArchiveForSetup(new File(openDJArchive));
    }

    private static SetupConsole sampleConsole() {
        return new SetupConsole() {
            @Override
            public void println() {
                System.out.println();
            }

            @Override
            public void handleInfo(LocalizableMessage localizableMessage) {
                System.out.println(localizableMessage);
            }

            @Override
            public void handleStartTask(LocalizableMessage task) {
                System.out.print(task + ".....");
            }

            @Override
            public void successEndTask() {
                System.out.println(" Done");
            }

            @Override
            public void handleError(LocalizableMessage message) {
```

```
                System.err.println(message);
            }
        };
    }

    private static void printUsageAndExit() {
        System.err.println(
                "Usage: OPENDJ_ARCHIVE SERVER_ROOT_DIR LDAP_PORT ADMIN_PORT (with_sample_backend |
 with_profiles)");
        System.exit(1);
    }

    private static int parseIntOrDefault(final String string, final int defaultValue) {
        try {
            return Integer.parseInt(string);
        } catch (final NumberFormatException e) {
            return defaultValue;
        }
    }

    private SetupServer() {
        // Not used.
    }
}
```

Starting and Stopping an Embedded Server

After completing the steps described in "Setting Up an Embedded Server", try the server start and stop sample, listed in "Start and Stop an Embedded Server":

To Start and Stop an Embedded Server

1. Configure the IDE to run the `StartStopServer.main()` method.

   You must provide the full path where embedded server was installed, such as `opendj/opendj-embedded-server-examples/target/opendj`.

   This is the sample `serverRootDir` argument.

2. (Optional) Set breakpoints if you want to step through the code.

3. Run the sample.

   The sample starts the server in the location you specified, and waits for you to enter **Ctrl-C**:

   - It creates an embedded directory server object with the `manageEmbeddedDirectoryServerForRestrictedOps()` method.

     This method creates an embedded server object for performing operations that do not require connection parameters. It takes as arguments configuration parameters, and the output streams where the server can display output and errors.

- It starts the server with the `EmbeddedDirectoryServer.start()` method.
- If successful, the sample displays the following output, waiting for your input to stop the server:

```
Starting the server...
Type Ctrl-C to stop the server
```

- When you type **Ctrl-C**, it stops the server with the `EmbeddedDirectoryServer.stop()` method.

If you prefer to run the samples directly without an IDE, execute the file `opendj-embedded-server-examples/README`.

Start and Stop an Embedded Server

```
/*
 * Copyright 2016-2018 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */

package org.forgerock.opendj.examples;

import static org.forgerock.opendj.server.embedded.ConfigParameters.configParams;
import static org.forgerock.opendj.server.embedded.EmbeddedDirectoryServer.*;

import java.nio.file.Path;
import java.nio.file.Paths;

import org.forgerock.i18n.LocalizableMessage;
import org.forgerock.opendj.server.embedded.EmbeddedDirectoryServer;
import org.forgerock.opendj.server.embedded.EmbeddedDirectoryServerException;

/**
 * Start and stop a server that is already installed.
 */
public final class StartStopServer {

    /**
     * Main method.
     * <p>
     * The server is started, and this program waits for a Control-C on the terminal to stop the server.
     *
     * @param args
     *            The command line arguments: serverRootDir
     * @throws EmbeddedDirectoryServerException
     *          If an error occurs
     */
    public static void main(final String[] args) throws EmbeddedDirectoryServerException {
        if (args.length != 1) {
            System.err.println("Usage: serverRootDir");
            System.exit(1);
        }
        final Path serverRootDir = Paths.get(args[0]);
```

```
        final EmbeddedDirectoryServer server =
               manageEmbeddedDirectoryServerForRestrictedOps(
                       configParams()
                           .serverRootDirectory(serverRootDir)
                           .configurationFile(serverRootDir.resolve("config").resolve("config.ldif")),
                           System.out,
                           System.err);

        Runtime.getRuntime().addShutdownHook(new Thread() {
            @Override
            public void run() {
                System.out.println("Shutting down ...");
                server.stop(StartStopServer.class.getName(),
                        LocalizableMessage.raw("Stopped after receiving Control-C"));
            }
        });

        System.out.println("Starting the server...");
        server.start();
        System.out.println("Type Ctrl-C to stop the server");
    }

    private StartStopServer() {
        // Not used.
    }
}
```

Configuring an Embedded Server

After completing the steps described in "Setting Up an Embedded Server", try the server configuration sample, listed in "Start and Stop Embedded Server":

To Configure an Embedded Server

1. Configure the IDE to run the `ConfigureServer.main()` method.

   Provide at minimum the following required arguments:

   1. The full path where the sample class should install the server, such as `opendj/opendj-embedded-server-examples/target/opendj`.

      This is the sample `serverRootDir` argument.

   2. A new base DN to add to the `userRoot` backend, such as `dc=example,dc=com`.

      This is the sample `newBaseDn` argument.

2. (Optional) Set breakpoints if you want to step through the code.

3. Run the sample.

The sample sets base DN you specify as the base DN of the `userRoot` backend:

- It creates an embedded directory server object with the `manageEmbeddedDirectoryServer()` method.
- It gets an object to access the server configuration with the `EmbeddedDirectoryServer.getConfiguration()` method.
- It uses the configuration object to read the base DNs in the `userRoot` backend, and to set your new base DN as the base DN for the backend.

  Notice the `commit()` method that applies the configuration.
- If successful, the sample displays the following output:

```
The current base Dn(s) of the user backend are: [o=example]
The base Dn of the user backend has been set to: dc=example,dc=com
```

If you prefer to run the samples directly without an IDE, execute the file `opendj-embedded-server-examples/README`.

Start and Stop Embedded Server

```
/*
 * Copyright 2016-2018 ForgeRock AS. All Rights Reserved
 *
 * Use of this code requires a commercial software license with ForgeRock AS.
 * or with one of its affiliates. All use shall be exclusively subject
 * to such license between the licensee and ForgeRock AS.
 */
package org.forgerock.opendj.examples;

import static org.forgerock.opendj.config.ValueOrExpression.newValue;
import static org.forgerock.opendj.server.embedded.ConfigParameters.configParams;
import static org.forgerock.opendj.server.embedded.ConnectionParameters.connectionParams;
import static org.forgerock.opendj.server.embedded.EmbeddedDirectoryServer.manageEmbeddedDirectoryServer;
import static java.util.Arrays.asList;

import java.io.IOException;
import java.nio.file.Path;
import java.nio.file.Paths;

import org.forgerock.opendj.config.AdminException;
import org.forgerock.opendj.config.client.ManagementContext;
import org.forgerock.opendj.ldap.Dn;
import org.forgerock.opendj.server.config.client.BackendCfgClient;
import org.forgerock.opendj.server.config.client.PluggableBackendCfgClient;
import org.forgerock.opendj.server.embedded.EmbeddedDirectoryServer;
import org.forgerock.opendj.server.embedded.EmbeddedDirectoryServerException;

/**
 * Provides an example of read and update of the configuration of a server that
 * is already installed.
 * <p>
 * The server may be running or not.
```

```
 */
public final class ConfigureServer {

    /**
     * Main method.
     * <p>
     * Read the current base Dn of user backend and then change it
     * to the one provided as argument.
     *
     * @param args
     *            The command line arguments: serverRootDir newBaseDn [ldapPort]
     * @throws Exception
     *          If an error occurs
     */
    public static void main(final String[] args) throws Exception {
        if (args.length != 2 && args.length != 3) {
            System.err.println("Usage: serverRootDir newBaseDn [ldapPort]");
            System.exit(1);
        }
        final Path serverRootDir = Paths.get(args[0]);
        final String newBaseDn = args[1];
        final int ldapPort = args.length > 2 ? Integer.parseInt(args[2]) : 1500;

        EmbeddedDirectoryServer server =
                manageEmbeddedDirectoryServer(
                        configParams()
                            .serverRootDirectory(serverRootDir)
                            .configurationFile(serverRootDir.resolve("config").resolve("config.ldif")),
                        connectionParams()
                            .hostName("localhost")
                            .ldapPort(ldapPort)
                            .bindDn("cn=Directory Manager")
                            .bindPassword("password"),
                        System.out,
                        System.err);

        // read the current base DN(s) of user backend and update it
        try (ManagementContext config = server.getConfiguration()) {
            final BackendCfgClient userRoot = config.getRootConfiguration().getBackend("userRoot");
            final PluggableBackendCfgClient cfg = (PluggableBackendCfgClient) userRoot;
            System.out.println("The current base Dn(s) of the user backend are: " + cfg.getBaseDn());
            cfg.setBaseDn(asList(newValue(Dn.valueOf(newBaseDn))));
            cfg.commit();
            System.out.println("The base Dn of the user backend has been set to: " + newBaseDn);
        } catch (AdminException | IOException | EmbeddedDirectoryServerException e) {
            System.err.println("A problem occurred when reading/updating configuration: " + e.toString());
        }
    }

    private ConfigureServer() {
        // Not used.
    }
}
```

Chapter 11

LDAP Result Codes

An operation result code as defined in RFC 4511 section 4.1.9 is used to indicate the final status of an operation. If a server detects multiple errors for an operation, only one result code is returned. The server should return the result code that best indicates the nature of the error encountered. Servers may return substituted result codes to prevent unauthorized disclosures.

OpenDJ LDAP Result Codes

Result Code	Name	Description
-1	Undefined	The result code that should only be used if the actual result code has not yet been determined. Despite not being a standard result code, it is an implementation of the null object design pattern for this type.
0	Success	The result code that indicates that the operation completed successfully.
1	Operations Error	The result code that indicates that the operation is not properly sequenced with relation to other operations (of same or different type). For example, this code is returned if the client attempts to StartTLS [RFC4346] while there are other uncompleted operations or if a TLS layer was already installed.
2	Protocol Error	The result code that indicates that the client sent a malformed or illegal request to the server.
3	Time Limit Exceeded	The result code that indicates that a time limit was exceeded while attempting to process the request.
4	Size Limit Exceeded	The result code that indicates that a size limit was exceeded while attempting to process the request.
5	Compare False	The result code that indicates that the attribute value assertion included in a compare request did not match the targeted entry.
6	Compare True	The result code that indicates that the attribute value assertion included in a compare request did match the targeted entry.
7	Authentication Method Not Supported	The result code that indicates that the requested authentication attempt failed because it referenced an invalid SASL mechanism.

Result Code	Name	Description
8	Strong Authentication Required	The result code that indicates that the requested operation could not be processed because it requires that the client has completed a strong form of authentication.
10	Referral	The result code that indicates that a referral was encountered. Strictly speaking this result code should not be exceptional since it is considered as a "success" response. However, referrals should occur rarely in practice and, when they do occur, should not be ignored since the application may believe that a request has succeeded when, in fact, nothing was done.
11	Administrative Limit Exceeded	The result code that indicates that processing on the requested operation could not continue because an administrative limit was exceeded.
12	Unavailable Critical Extension	The result code that indicates that the requested operation failed because it included a critical extension that is unsupported or inappropriate for that request.
13	Confidentiality Required	The result code that indicates that the requested operation could not be processed because it requires confidentiality for the communication between the client and the server.
14	SASL Bind in Progress	The result code that should be used for intermediate responses in multi-stage SASL bind operations.
16	No Such Attribute	The result code that indicates that the requested operation failed because it targeted an attribute or attribute value that did not exist in the specified entry.
17	Undefined Attribute Type	The result code that indicates that the requested operation failed because it referenced an attribute that is not defined in the server schema.
18	Inappropriate Matching	The result code that indicates that the requested operation failed because it attempted to perform an inappropriate type of matching against an attribute.
19	Constraint Violation	The result code that indicates that the requested operation failed because it would have violated some constraint defined in the server.
20	Attribute or Value Exists	The result code that indicates that the requested operation failed because it would have resulted in a conflict with an existing attribute or attribute value in the target entry.
21	Invalid Attribute Syntax	The result code that indicates that the requested operation failed because it violated the syntax for a specified attribute.

Result Code	Name	Description
32	No Such Entry	The result code that indicates that the requested operation failed because it referenced an entry that does not exist.
33	Alias Problem	The result code that indicates that the requested operation failed because it attempted to perform an illegal operation on an alias.
34	Invalid DN Syntax	The result code that indicates that the requested operation failed because it would have resulted in an entry with an invalid or malformed DN.
36	Alias Dereferencing Problem	The result code that indicates that a problem was encountered while attempting to dereference an alias for a search operation.
48	Inappropriate Authentication	The result code that indicates that an authentication attempt failed because the requested type of authentication was not appropriate for the targeted entry.
49	Invalid Credentials	The result code that indicates that an authentication attempt failed because the user did not provide a valid set of credentials.
50	Insufficient Access Rights	The result code that indicates that the client does not have sufficient permission to perform the requested operation.
51	Busy	The result code that indicates that the server is too busy to process the requested operation. This is a transient error which means the operation can safely be retried.
52	Unavailable	The result code that indicates that either the entire server or one or more required resources were not available for use in processing the request. This is a transient error which means the operation can safely be retried.
53	Unwilling to Perform	The result code that indicates that the server is unwilling to perform the requested operation.
54	Loop Detected	The result code that indicates that a referral or chaining loop was detected while processing the request.
60	Sort Control Missing	The result code that indicates that a search request included a VLV request control without a server-side sort control.
61	Offset Range Error	The result code that indicates that a search request included a VLV request control with an invalid offset.

Result Code	Name	Description
64	Naming Violation	The result code that indicates that the requested operation failed because it would have violated the server's naming configuration.
65	Object Class Violation	The result code that indicates that the requested operation failed because it would have resulted in an entry that violated the server schema.
66	Not Allowed on Non-Leaf	The result code that indicates that the requested operation is not allowed for non-leaf entries.
67	Not Allowed on RDN	The result code that indicates that the requested operation is not allowed on an RDN attribute.
68	Entry Already Exists	The result code that indicates that the requested operation failed because it would have resulted in an entry that conflicts with an entry that already exists.
69	Object Class Modifications Prohibited	The result code that indicates that the operation could not be processed because it would have modified the objectclasses associated with an entry in an illegal manner.
71	Affects Multiple DSAs	The result code that indicates that the operation could not be processed because it would impact multiple DSAs or other repositories.
76	Virtual List View Error	The result code that indicates that the operation could not be processed because there was an error while processing the virtual list view control.
80	Other	The result code that should be used if no other result code is appropriate.
81	Server Connection Closed	The client-side result code that indicates that the server is down. This is for client-side use only and should never be transferred over protocol. This is a transient error which means the operation can be retried.
82	Local Error	The client-side result code that indicates that a local error occurred that had nothing to do with interaction with the server. This is for client-side use only and should never be transferred over protocol.
83	Encoding Error	The client-side result code that indicates that an error occurred while encoding a request to send to the server. This is for client-side use only and should never be transferred over protocol.
84	Decoding Error	The client-side result code that indicates that an error occurred while decoding a response from the server. This is for client-side use only and should never be transferred over protocol.
85	Client-Side Timeout	The client-side result code that indicates that the client did not receive an expected response in a timely

Result Code	Name	Description
		manner. This is for client-side use only and should never be transferred over protocol. This is a transient error which means the operation can be retried.
86	Unknown Authentication Mechanism	The client-side result code that indicates that the user requested an unknown or unsupported authentication mechanism. This is for client-side use only and should never be transferred over protocol.
87	Filter Error	The client-side result code that indicates that the filter provided by the user was malformed and could not be parsed. This is for client-side use only and should never be transferred over protocol.
88	Cancelled by User	The client-side result code that indicates that the user cancelled an operation. This is for client-side use only and should never be transferred over protocol.
89	Parameter Error	The client-side result code that indicates that there was a problem with one or more of the parameters provided by the user. This is for client-side use only and should never be transferred over protocol.
90	Out of Memory	The client-side result code that indicates that the client application was not able to allocate enough memory for the requested operation. This is for client-side use only and should never be transferred over protocol.
91	Connect Error	The client-side result code that indicates that the client was not able to establish a connection to the server. This is for client-side use only and should never be transferred over protocol. This is a transient error which means the operation can be retried.
92	Operation Not Supported	The client-side result code that indicates that the user requested an operation that is not supported. This is for client-side use only and should never be transferred over protocol.
93	Control Not Found	The client-side result code that indicates that the client expected a control to be present in the response from the server but it was not included. This is for client-side use only and should never be transferred over protocol.
94	No Results Returned	The client-side result code that indicates that the requested single entry search operation or read operation failed because the Directory Server did not return any matching entries. This is for client-side use only and should never be transferred over protocol.
95	Unexpected Results Returned	The client-side result code that the requested single entry search operation or read operation failed because the Directory Server returned multiple matching entries (or search references) when only a

Result Code	Name	Description
		single matching entry was expected. This is for client-side use only and should never be transferred over protocol.
96	Referral Loop Detected	The client-side result code that indicates that the client detected a referral loop caused by servers referencing each other in a circular manner. This is for client-side use only and should never be transferred over protocol.
97	Referral Hop Limit Exceeded	The client-side result code that indicates that the client reached the maximum number of hops allowed when attempting to follow a referral (i.e., following one referral resulted in another referral which resulted in another referral and so on). This is for client-side use only and should never be transferred over protocol.
118	Canceled	The result code that indicates that a cancel request was successful, or that the specified operation was canceled.
119	No Such Operation	The result code that indicates that a cancel request was unsuccessful because the targeted operation did not exist or had already completed.
120	Too Late	The result code that indicates that a cancel request was unsuccessful because processing on the targeted operation had already reached a point at which it could not be canceled.
121	Cannot Cancel	The result code that indicates that a cancel request was unsuccessful because the targeted operation was one that could not be canceled.
122	Assertion Failed	The result code that indicates that the filter contained in an assertion control failed to match the target entry.
123	Authorization Denied	The result code that should be used if the server will not allow the client to use the requested authorization.
16,654	No Operation	The result code that should be used if the server did not actually complete processing on the associated operation because the request included the LDAP No-Op control.

Appendix A. Getting Support

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.forgerock.com.

ForgeRock has staff members around the globe who support our international customers and partners. For details on ForgeRock's support offering, including support plans and service level agreements (SLAs), visit https://www.forgerock.com/support.

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

  While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

Glossary

Abandon operation	LDAP operation to stop processing of a request in progress, after which the server drops the connection without a reply to the client application.
Access control	Control to grant or to deny access to a resource.
Access control instruction (ACI)	Instruction added as a directory entry attribute for fine-grained control over what a given user or group member is authorized to do in terms of LDAP operations and access to user data. ACIs are implemented independently from privileges, which apply to administrative operations. See also Privilege.
Access control list (ACL)	An access control list connects a user or group of users to one or more security entitlements. For example, users in group sales are granted the entitlement read-only to some financial data.
`access` log	Server log tracing the operations the server processes including timestamps, connection information, and information about the operation itself.
Account lockout	The act of making an account temporarily or permanently inactive after successive authentication failures.
Active user	A user that has the ability to authenticate and use the services, having valid credentials.
Add operation	LDAP operation to add a new entry or entries to the directory.

Anonymous	A user that does not need to authenticate, and is unknown to the system.
Anonymous bind	A bind operation using simple authentication with an empty DN and an empty password, allowing anonymous access such as reading public information.
Approximate index	Index is used to match values that "sound like" those provided in the filter.
Attribute	Properties of a directory entry, stored as one or more key-value pairs. Typical examples include the common name (`cn`) to store the user's full name and variations of the name, user ID (`uid`) to store a unique identifier for the entry, and `mail` to store email addresses.
`audit` log	Type of access log that dumps changes in LDIF.
Authentication	The process of verifying who is requesting access to a resource; the act of confirming the identity of a principal.
Authorization	The process of determining whether access should be granted to an individual based on information about that individual; the act of determining whether to grant or to deny a principal access to a resource.
Backend	Repository that stores directory data. Different implementations with different capabilities exist.
Binary copy	Binary backup archive of one directory server that can be restored on another directory server.
Bind operation	LDAP authentication operation to determine the client's identity in LDAP terms, the identity which is later used by the server to authorize (or not) access to directory data that the client wants to lookup or change.
Branch	The distinguished name (DN) of a non-leaf entry in the Directory Information Tree (DIT), and also that entry and all its subordinates taken together. Some administrative operations allow you to include or exclude branches by specifying the DN of the branch. See also Suffix.
Collective attribute	A standard mechanism for defining attributes that appear on all the entries in a particular subtree.
Compare operation	LDAP operation to compare a specified attribute value with the value stored on an entry in the directory.

Control	Information added to an LDAP message to further specify how an LDAP operation should be processed. DS supports many LDAP controls.
Database cache	Memory space set aside to hold database content.
`debug` log	Server log tracing details needed to troubleshoot a problem in the server.
Delete operation	LDAP operation to remove an existing entry or entries from the directory.
Directory	A directory is a network service which lists participants in the network such as users, computers, printers, and groups. The directory provides a convenient, centralized, and robust mechanism for publishing and consuming information about network participants.
Directory hierarchy	A directory can be organized into a hierarchy in order to make it easier to browse or manage. Directory hierarchies normally represent something in the physical world, such as organizational hierarchies or physical locations. For example, the top level of a directory may represent a company, the next level down divisions, the next level down departments, and down the hierarchy. Alternately, the top level may represent the world, the next level down countries, next states or provinces, and next cities.
Directory Information Tree (DIT)	A set of directory entries organized hierarchically in a tree structure, where the vertices are the entries and the arcs between vertices define relationships between entries
Directory manager	Default directory superuser who has privileges to do full administration of the DS server, including bypassing access control evaluation, changing access controls, and changing administrative privileges. See also Superuser.
Directory object	A directory object is an item in a directory. Example objects include users, user groups, computers, and more. Objects may be organized into a hierarchy and contain identifying attributes. See also Entry.
Directory proxy server	Server that forwards LDAP requests to remote directory servers. A standalone directory proxy server does not store user data. See also Directory server.
Directory server	Server application for centralizing information about network participants. A highly available directory service consists of multiple directory servers configured to replicate directory data. See also Directory, Replication.

Directory Services Markup Language (DSML)	Standard language to access directory services using XML. DMSL v1 defined an XML mapping of LDAP objects, while DSMLv2 maps the LDAP Protocol and data model to XML.
Distinguished name (DN)	Fully qualified name for a directory entry, such as `uid=bjensen, ou=People,dc=example,dc=com`, built by concatenating the entry RDN (`uid=bjensen`) with the DN of the parent entry (`ou=People,dc=example, dc=com`).
Domain	A replication domain consists of several directory servers sharing the same synchronized set of data. The base DN of a replication domain specifies the base DN of the replicated data.
DSML gateway	Standalone web application that translates DSML requests from client applications to LDAP requests to a directory service, and LDAP responses from a directory service to DSML responses to client applications.
Dynamic group	Group that specifies members using LDAP URLs.
Entry	As generic and hierarchical data stores, directories always contain different kinds of entries, either nodes (or containers) or leaf entries. An entry is an object in the directory, defined by one of more object classes and their related attributes. At startup, DS servers report the number of entries contained in each suffix.
Entry cache	Memory space set aside to hold frequently accessed, large entries, such as static groups.
Equality index	Index used to match values that correspond exactly (though generally without case sensitivity) to the value provided in the search filter.
`errors` log	Server log tracing server events, error conditions, and warnings, categorized and identified by severity.
Export	Save directory data in an LDIF file.
Extended operation	Additional LDAP operation not included in the original standards. DS servers support several standard LDAP extended operations.
Extensible match index	Index for a matching rule other than approximate, equality, ordering, presence, substring or VLV, such as an index for generalized time.
External user	An individual that accesses company resources or services but is not working for the company. Typically a customer or partner.
Etime	Elapsed time within the server to process a request, starting from the moment the decoded operation is available to be processed by a worker thread.

Filter	An LDAP search filter is an expression that the server uses to find entries that match a search request, such as `(mail=*@example.com)` to match all entries having an email address in the example.com domain.
Group	Entry identifying a set of members whose entries are also in the directory.
Idle time limit	Defines how long DS allows idle connections to remain open.
Import	Read in and index directory data from an LDIF file.
Inactive user	An entry in the directory that once represented a user but which is now no longer able to be authenticated.
Index	Directory server backend feature to allow quick lookup of entries based on their attribute values. See also Approximate index, Equality index, Extensible match index, Ordering index, Presence index, Substring index, Virtual list view (VLV) index, Index entry limit.
Index entry limit	When the number of entries that an index key points to exceeds the index entry limit, DS stops maintaining the list of entries for that index key.
Internal user	An individual who works within the company either as an employee or as a contractor.
LDAP Data Interchange Format (LDIF)	Standard, portable, text-based representation of directory content. See RFC 2849.
LDAP URL	LDAP Uniform Resource Locator such as `ldap://directory.example.com:389/dc=example,dc=com??sub?(uid=bjensen)`. See RFC 2255.
LDAPS	LDAP over SSL.
Lightweight Directory Access Protocol (LDAP)	A simple and standardized network protocol used by applications to connect to a directory, search for objects and add, edit or remove objects. See RFC 4510.
Lookthrough limit	Defines the maximum number of candidate entries DS considers when processing a search.
Matching rule	Defines rules for performing matching operations against assertion values. Matching rules are frequently associated with an attribute syntax and are used to compare values according to that syntax. For example, the `distinguishedNameEqualityMatch` matching rule can be used to determine whether two DNs are equal and can ignore unnecessary spaces around commas and equal signs, differences in capitalization in attribute names, and other discrepancies.

Modify DN operation	LDAP modification operation to request that the server change the distinguished name of an entry.
Modify operation	LDAP modification operation to request that the server change one or more attributes of an entry.
Naming context	Base DN under which client applications can look for user data.
Object class	Identifies entries that share certain characteristics. Most commonly, an entry's object classes define the attributes that must and may be present on the entry. Object classes are stored on entries as values of the `objectClass` attribute. Object classes are defined in the directory schema, and can be abstract (defining characteristics for other object classes to inherit), structural (defining the basic structure of an entry, one structural inheritance per entry), or auxiliary (for decorating entries already having a structural object class with other required and optional attributes).
Object identifier (OID)	String that uniquely identifies an object, such as `0.9.2342.19200300.100.1.1` for the user ID attribute or `1.3.6.1.4.1.1466.115.121.1.15` for `DirectoryString` syntax.
Operational attribute	An attribute that has a special (operational) meaning for the server, such as `pwdPolicySubentry` or `modifyTimestamp`.
Ordering index	Index used to match values for a filter that specifies a range.
Password policy	A set of rules regarding what sequence of characters constitutes an acceptable password. Acceptable passwords are generally those that would be too difficult for another user or an automated program to guess and thereby defeat the password mechanism. Password policies may require a minimum length, a mixture of different types of characters (lowercase, uppercase, digits, punctuation marks, and other characters), avoiding dictionary words or passwords based on the user's name, and other attributes. Password policies may also require that users not reuse old passwords and that users change their passwords regularly.
Password reset	Password change performed by a user other than the user who owns the entry.
Password storage scheme	Mechanism for encoding user passwords stored on directory entries. DS implements a number of password storage schemes.
Password validator	Mechanism for determining whether a proposed password is acceptable for use. DS implements a number of password validators.
Plugin	Java library with accompanying configuration that implements a feature through processing that is not essential to the core operation of DS servers.

	As the name indicates, plugins can be plugged in to an installed server for immediate configuration and use without recompiling the server. DS servers invoke plugins at specific points in the lifecycle of a client request. The DS configuration framework lets directory administrators manage plugins with the same tools used to manage the server.
Presence index	Index used to match the fact that an attribute is present on the entry, regardless of the value.
Principal	Entity that can be authenticated, such as a user, a device, or an application.
Privilege	Server configuration settings controlling access to administrative operations such as exporting and importing data, restarting the server, performing password reset, and changing the server configuration. Privileges are implemented independently from access control instructions (ACI), which apply to LDAP operations and user data. See also Access control instruction (ACI).
Referential integrity	Ensuring that group membership remains consistent following changes to member entries.
`referint` log	Server log tracing referential integrity events, with entries similar to the errors log.
Referral	Reference to another directory location, which can be another directory server running elsewhere or another container on the same server, where the current operation can be processed.
Relative distinguished name (RDN)	Initial portion of a DN that distinguishes the entry from all other entries at the same level, such as `uid=bjensen` in `uid=bjensen,ou=People,dc=example,dc=com`.
Replica	Directory server this is configured to use replication.
Replication	Data synchronization that ensures all directory servers participating eventually share a consistent set of directory data.
`replication` log	Server log tracing replication events, with entries similar to the errors log.
Replication server	Server dedicated to transmitting replication messages. A standalone replication server does not store user data.
REST to LDAP gateway	Standalone web application that translates RESTful HTTP requests from client applications to LDAP requests to directory services, and

	LDAP responses from directory services to HTTP responses to client applications.
Root DSE	The directory entry with distinguished name "" (empty string), where DSE is an acronym for DSA-Specific Entry. DSA is an acronym for Directory Server Agent, a single directory server. The root DSE serves to expose information over LDAP about what the directory server supports in terms of LDAP controls, auth password schemes, SASL mechanisms, LDAP protocol versions, naming contexts, features, LDAP extended operations, and other information.
Schema	LDAP schema defines the object classes, attributes types, attribute value syntaxes, matching rules and other constrains on entries held by the directory server.
Search filter	See Filter.
Search operation	LDAP lookup operation where a client requests that the server return entries based on an LDAP filter and a base DN under which to search.
Simple authentication	Bind operation performed with a user's entry DN and user's password. Use simple authentication only if the network connection is secure.
Size limit	Sets the maximum number of entries returned for a search.
Static group	Group that enumerates member entries.
Subentry	An entry, such as a password policy entry, that resides with the user data but holds operational data, and is not visible in search results unless explicitly requested.
Substring index	Index used to match values specified with wildcards in the filter.
Suffix	The distinguished name (DN) of a root entry in the Directory Information Tree (DIT), and also that entry and all its subordinates taken together as a single object of administrative tasks such as export, import, indexing, and replication.
Superuser	User with privileges to perform unconstrained administrative actions on DS server. This account is analogous to the UNIX `root` and Windows `Administrator` accounts. Superuser privileges include the following: • `bypass-acl`: The holder is not subject to access control. • `privilege-change`: The holder can edit administrative privileges. • `proxied-auth`: The holder can make requests on behalf of another user, including directory superusers.

	The default superuser is `cn=Directory Manager`. You can create additional superuser accounts, each with different administrative privileges. See also Directory manager, Privilege.
Task	Mechanism to provide remote access to server administrative functions. DS software supports tasks to back up and restore backends, to import and export LDIF files, and to stop and restart the server.
Time limit	Defines the maximum processing time DS devotes to a search operation.
Unbind operation	LDAP operation to release resources at the end of a session.
Unindexed search	Search operation for which no matching index is available. If no indexes are applicable, then the directory server potentially has to go through all entries to look for candidate matches. For this reason, the `unindexed-search` privilege, which allows users to request searches for which no applicable index exists, is reserved for the directory manager by default.
User	An entry that represents an individual that can be authenticated through credentials contained or referenced by its attributes. A user may represent an internal user or an external user, and may be an active user or an inactive user.
User attribute	An attribute for storing user data on a directory entry such as `mail` or `givenname`.
Virtual attribute	An attribute with dynamically generated values that appear in entries but are not persistently stored in the backend.
Virtual directory	An application that exposes a consolidated view of multiple physical directories over an LDAP interface. Consumers of the directory information connect to the virtual directory's LDAP service. Behind the scenes, requests for information and updates to the directory are sent to one or more physical directories where the actual information resides. Virtual directories enable organizations to create a consolidated view of information that for legal or technical reasons cannot be consolidated into a single physical copy.
Virtual list view (VLV) index	Browsing index designed to help the directory server respond to client applications that need, for example, to browse through a long list of results a page at a time in a GUI.
Virtual static group	DS group that lets applications see dynamic groups as what appear to be static groups.

X.500	A family of standardized protocols for accessing, browsing and maintaining a directory. X.500 is functionally similar to LDAP, but is generally considered to be more complex, and has consequently not been widely adopted.