LDAP User Guide

/ Directory Services 7

Latest update: 7.0.2

Mark Craig

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

Abstract

Guide to the ForgeRock® Directory Services LDAP features and tools.

Table of Contents

Overview

This guide shows you how to use DS LDAP features and command-line tools.

Quick Start

 Tools Find and run DS command-line tools.	 Authentication Understand LDAP authentication (binds).	 Searches Use LDAP searches to look up information in the directory.
 Comparisons Use LDAP comparisons to check for attribute value matches.	 Updates Use LDAP to add, modify, and delete directory data.	 Passwords Change your own password and administer passwords for other users.

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

Chapter 1
About DS Tools

Client Tools

- Add DS client command-line tools to your PATH:

  Bash

  ```
  $ export PATH=/path/to/opendj/bin:${PATH}
  ```

  PowerShell

  ```
  PS C:\path\to> $env:PATH += ";C:\path\to\opendj\bat"
  ```

- For reference information, use the `--help` option with any DS tool.

- All commands call Java programs. This means every command starts a JVM, so it takes longer to start than a native binary.

Command[a]	Description
addrate	Measure add and delete throughput and response time.
authrate	Measure bind throughput and response time.
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.
ldapcompare	Compare the attribute values you specify with those stored on entries in the directory.
ldapdelete	Delete entries from the directory.
ldapmodify	Modify the specified attribute values for the specified entries.
ldappasswordmodify	Modify user passwords.
ldapsearch	Search a branch of directory data for entries that match the LDAP filter you specify.
ldiffdiff	Display differences between two LDIF files, with the resulting output having LDIF format.
ldifmodify	Modify specified attribute values for specified entries in an LDIF file.
ldifsearch	Search a branch of data in LDIF for entries matching the LDAP filter you specify.

Command[a]	Description
makeldif	Generate directory data in LDIF based on templates that define how the data should appear. Also see makeldif-template.
modrate	Measure modification throughput and response time.
searchrate	Measure search throughput and response time.

[a] UNIX names for the commands. Equivalent Windows commands have `.bat` extensions.

Trusted Certificates

When a client tool initiates a secure connection to a server, the server presents its digital certificate. The tool must determine whether it trusts the server certificate and continues to negotiate a secure connection, or does not trust the server certificate and drops the connection. To trust the server certificate, the tool's truststore must contain the trusted certificate. The trusted certificate is a CA certificate, or the self-signed server certificate. The following table explains how the tools locate the truststore.

Truststore Option	Truststore Used
None	The default truststore, *`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. • In interactive mode, DS command-line tools prompt for approval to trust an unrecognized certificate, and whether to store it in the default truststore for future use. • In silent mode, the tools rely on the default truststore.

`-P {trustStorePath}`
 	
`--trustStorePath {trustStorePath}`
	Only the specified truststore is used.
	
	The tool fails with an error if the server certificate is not trusted.

Default Settings

You can set defaults in the `~/.opendj/tools.properties` file as in the following example:

```
hostname=localhost
port=1636
bindDN=uid=kvaughan,ou=People,dc=example,dc=com
useSsl=true
```

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

To override the settings, use the `--noPropertiesFile` option.

Chapter 2

Authentication (Binds)

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.

DS servers implement fine-grained access control for authorization. Authorization for an operation depends on who is requesting the operation. DS servers must authenticate the principal before making an authorization decision. In LDAP, the bind operation authenticates the principal.

Clients bind by providing a means to find their principal's entry, and credentials to check against the entry:

- In a simple bind operation, the client provides an LDAP name, usually the DN identifying its entry, and the corresponding password stored in the 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. LDAP servers may allow anonymous binds to read public information, such as root DSE attributes.

- Other bind mechanisms involve digital certificates, Kerberos tickets, or challenge response mechanisms that prove the client knows a password.

A user rarely knows, let alone enters, their DN. Instead, a user provides a client application with an identifying string stored in their entry, such as a user ID or an email address. The client application builds the DN directly from the user's identity string, or searches for the user entry based on the user's identity string to find the DN. The client application performs a simple bind with the resulting DN.

For example, suppose Babs Jensen enters her email address, `bjensen@example.com`, and password. 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 extract the user ID `bjensen` from the address, then build the corresponding DN, `uid=bjensen,ou=people,dc=example,dc=com`, without a lookup.

Identity Mappers

When the mapping from the user identifier to the DN is known, DS servers can use an *identity mapper* to do the translation. 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 map authorization IDs to DNs for 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 example, Babs Jensen has access to read the hashed value of her password. Notice the authentication ID is her user ID, `u:bjensen`, rather than the DN of her entry:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 --saslOption mech=PLAIN \
 --saslOption authid=u:bjensen \
 --bindPassword hifalutin \
 "(cn=Babs Jensen)" \
 userPassword
dn: uid=bjensen,ou=People,dc=example,dc=com
userPassword: {PBKDF2-HMAC-SHA256}10000:<hash>
```

The "Exact Match Identity Mapper" searches for a match between the string (here, `bjensen`), and the value of a specified attribute (by default, the `uid` attribute). By default, the identity mapper searches all public naming contexts local to the server. If duplicate entries exist, or if the required indexes are not available for all backends, this behavior can be restricted using the `match-base-dn` property.

You can configure multiple identity mappers, if necessary. When resolving the identity, the server uses the first identity mapper that finds a match. If multiple identity mappers match different entries, however, then the server returns LDAP error code 19, Constraint Violation.

If you know that users are entering their email addresses, you could create an exact match identity mapper for email addresses, then use that for PLAIN SASL authentication:

+ *Show example*

```
$ dsconfig \
 create-identity-mapper \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --mapper-name "Email Mapper" \
 --type exact-match \
 --set match-attribute:mail \
 --set enabled:true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --no-prompt
$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --hostname localhost \
```

```
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name PLAIN \
 --set identity-mapper:"Email Mapper" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --no-prompt
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 --saslOption mech=PLAIN \
 --saslOption authid=u:bjensen@example.com \
 --bindPassword hifalutin \
 "(cn=Babs Jensen)" \
 userPassword
dn: uid=bjensen,ou=People,dc=example,dc=com
userPassword: {PBKDF2-HMAC-SHA256}10000:<hash>
```

A "Regular Expression Identity Mapper" uses a regular expression to extract a substring from the string provided. The server searches for a match between the substring and the value of a specified attribute. When an email address is *user ID* + @ + *domain*, you can use the default regular expression identity mapper in the same way as the email mapper in the example above. 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 localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --handler-name PLAIN \
 --set identity-mapper:"Regular Expression" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --no-prompt
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 --saslOption mech=PLAIN \
 --saslOption authid=u:bjensen@example.com \
 --bindPassword hifalutin \
 "(cn=Babs Jensen)" \
 userPassword
dn: uid=bjensen,ou=People,dc=example,dc=com
userPassword: {PBKDF2-HMAC-SHA256}10000:<hash>
```

Use 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 `javax.util.regex.Pattern` regular expression.

Like the exact match identity mapper, the regular expression identity mapper searches all public naming contexts local to the server by default. If duplicate entries exist, this behavior can be restricted using the `match-base-dn` property.

Chapter 3
LDAP Search

Note

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see "Learn About the Evaluation Setup Profile" in the *Installation Guide*.

Searching the directory is like 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 phone book has only one index (alphabetical order by name), the directory has many indexes. When performing a search, you specify which attributes to use, and the server derives the corresponding indexes.

The phone book might be divided into white pages for residential subscribers and yellow pages for businesses. If you look up an individual's phone number, you limit your search to the white pages. Directory services divide entries in various ways. For example, they can store organizations and groups in different locations from user entries or printer accounts. When searching the directory, you therefore also specify where to search.

The **ldapsearch** command requires arguments for at least the 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 use `ou=Printers,dc=example,dc=com`. In the `GNB00` office, you could look up a printer as shown in the following example:

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

In the example above, the LDAP filter matches printer entries where the `printerLocation` attribute is equal to `GNB00`.

You also specify the host and port to access directory services, and the protocol to use, such as LDAP or LDAPS. If the directory service does not allow anonymous access to the data you want to search, you supply credentials, such as a username and password, or a public key certificate. You can optionally specify a list of attributes to return. If you do not specify attributes, then the search returns all user attributes for the entry.

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

Simple LDAP Filter

The following example searches for entries with user IDs (`sn`) equal to `hall`, returning only DNs and user ID values:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(sn=hall)" \
 uid
dn: uid=ahall,ou=People,dc=example,dc=com
uid: ahall

dn: uid=bhal2,ou=People,dc=example,dc=com
uid: bhal2

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

Complex LDAP Filter

The following example returns entries with `sn` equal to `jensen` for users located in San Francisco:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --baseDN ou=people,dc=example,dc=com \
```

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

dn: uid=rjensen,ou=People,dc=example,dc=com
objectClass: person
objectClass: cos
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: posixAccount
objectClass: top
cn: Richard Jensen
description: Description on ou=People
sn: Jensen
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))`.

Return Operational Attributes

Operational attributes are returned only when explicitly requested. Use `+` in the attribute list after the filter to return all operational attributes, as in the following example:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --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.

Return Attributes of an Object Class

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

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
  @person
dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: person
objectClass: cos
objectClass: jsonObject
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: posixAccount
objectClass: top
cn: Barbara Jensen
cn: Babs Jensen
description: Original description
sn: Jensen
telephoneNumber: +1 408 555 1862
```

Approximate Match

DS servers support searches 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 "Approximate Index" in the *Configuration 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 localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --backend-name dsEvaluation \
 --index-name sn \
 --set index-type:approximate \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --no-prompt
$ rebuild-index \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 --index sn
```

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

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --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`.

Escape Characters in Filters

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 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(cn=\28A \5cgreat\5c name\2a\29)" \
 cn
dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
cn: (A \great\ name*)
```

Active Accounts

DS servers support extensible matching rules. Use a filter that specifies a matching rule OID that extends the matching operator.

DS servers support time-based matching rules for use with attributes that hold timestamp values:

Name: `relativeTimeOrderingMatch.gt`
OID: `1.3.6.1.4.1.26027.1.4.5`

Greater-than relative time matching rule for time-based searches.

Use this in a filter to match attributes with values greater than the current time +/- an offset.

The filter `(pwdExpirationTime:1.3.6.1.4.1.26027.1.4.5:=5d)` matches entries where the password expiration time is greater than the current time plus five days. In other words, entries whose passwords expire in more than five days.

Name: `relativeTimeOrderingMatch.lt`
OID: `1.3.6.1.4.1.26027.1.4.6`

Less-than relative time matching rule for time-based searches.

Use this in a filter to match attributes with values less than the current time +/- an offset.

The filter `(lastLoginTime:1.3.6.1.4.1.26027.1.4.6:=-4w)` matches entries where the last login time is less than the current time minus four weeks. In other words, accounts that have not been active in the last four weeks.

Name: `partialDateAndTimeMatchingRule`
OID: `1.3.6.1.4.1.26027.1.4.7`

Partial date and time matching rule for matching parts of dates in time-based searches.

The filter `(lastLoginTime:1.3.6.1.4.1.26027.1.4.7:=2020)` matches entries where the last login time was in 2020.

The following example defines a `lastLoginTime` attribute:

- The new attribute is an operational attribute (`USAGE directoryOperation`).

  When checking schema compliance, the server skips operational attributes. The server can therefore add operational attributes to an entry without changing the entry's object classes.

  Operational attributes hold information for the directory, rather than information targeting client applications. The server returns operational attributes only when explicitly requested, and client applications generally should not be able to modify them.

  By defining the `lastLoginTime` attribute as operational, you limit its visibility. You also help prevent client applications from modifying its value unless specifically allowed to.

- The new attribute has Generalized Time syntax (`SYNTAX 1.3.6.1.4.1.1466.115.121.1.24`).

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
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' )
EOF
```

Configure the applicable password policy to write the last login timestamp when a user authenticates.

The following command configures a subentry password policy. On successful authentication, the policy causes the server to write a timestamp in generalized time format to the user's `lastLoginTime` operational attribute:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: cn=Record last login,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: ds-pwp-password-policy
cn: Record last login
ds-pwp-password-attribute: userPassword
ds-pwp-default-password-storage-scheme: PBKDF2-HMAC-SHA256
ds-pwp-last-login-time-attribute: lastLoginTime
ds-pwp-last-login-time-format: yyyyMMddHH'Z'
subtreeSpecification: { base "ou=people" }
EOF
```

The `ds-pwp-last-login-time-format` setting must:

- Match the syntax of the `ds-pwp-last-login-time-attribute` attribute, which in this example is `GeneralizedTime`.
- Be a valid format string for the `java.text.SimpleDateFormat` class.

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

```
$ dsconfig \
 create-backend-index \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --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 \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --no-prompt
$ rebuild-index \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 --index lastLoginTime
```

Make sure you have some users who have authenticated recently:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=bjensen,ou=people,dc=example,dc=com \
 --bindPassword hifalutin \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 1.1

$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --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 13 weeks:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(lastLoginTime:1.3.6.1.4.1.26027.1.4.5:=-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 this year:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --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
```

Language Subtypes

DS servers support the language subtypes listed in "*Support for Languages and Locales*" in the *LDAP 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 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(cn=Frederique Dupont)" cn\;lang-fr
dn: uid=fdupont,ou=People,dc=example,dc=com
cn;lang-fr:: RnJlZMOpcmlxdWUgRHVwb250
$ base64 decode --encodedData RnJlZMOpcmlxdWUgRHVwb250
Fredérique 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

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 might not be permitted, depending on the attribute.	`"(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.
`>=`	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 with a `userPassword` attribute, 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	`(uid:dn:=bjensen)` matches entries with DN component `uid=bjensen`. `(lastLoginTime: 1.3.6.1.4.1.26027.1.4.5:=-13w)` matches entries with a last login time more recent than 13 weeks. Extensible match filters work with localized values. DS servers support 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

Operator	Definition	Example
	• Add `.5` for greater than • Add `.6` for substring	language subtype, such as `fr`, instead of the OID. `"(cn:dn:=My App)"` matches entries with `cn: My App` and DN component `cn=My App`.
`!`	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.

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" in the *HTTP User Guide*. This example depends on settings applied with the `ds-evaluation` setup profile.

The index lets you search with Common REST query filters. The following example finds the entry with a JSON attribute with `"access_token": "123"`:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --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"}
```

You can combine Common REST query filter syntax filters with other LDAP search filter to form complex filters, as demonstrated in "Complex LDAP Filter". For example, `(&(json=access_token eq '123')(mail=bjensen@example.com))`.

JSON Assertions

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 settings applied with the `ds-evaluation` setup profile.

Search for entries with a `jsonToken` attribute:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 '(jsonToken={"id":"HgAaB6xDhLom4JbM"})' \
 jsonToken
jsonToken: {"id":"HgAaB6xDhLom4JbM","scopes":["read","write"],"expires":"2018-01-10T10:08:34Z"}
```

Server-Side Sort

If permitted by the directory administrator, you can request that the server sort the search results. 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 results on the client side, or to browse results with a search that matches a virtual list view index, as demonstrated in "Virtual List View Index" in the *Configuration Guide*.

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

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDn uid=admin \
 --bindPassword password << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetcontrol = "PSearch")
 (version 3.0;acl "Allow Server-Side Sort for Kirsten Vaughan";
 allow (read)(userdn = "ldap:///uid=kvaughan,ou=People,dc=example,dc=com");)
EOF
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDn uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDn dc=example,dc=com \
 --sortOrder +sn \
 "(&(sn=*)(cn=babs*))" \
 cn
dn: uid=user.94643,ou=People,dc=example,dc=com
cn: Babs Bautista
```

```
dn: uid=user.81225,ou=People,dc=example,dc=com
cn: Babs Bawek

dn: uid=user.67807,ou=People,dc=example,dc=com
cn: Babs Baxter

dn: uid=user.54389,ou=People,dc=example,dc=com
cn: Babs Bayer

dn: uid=user.40971,ou=People,dc=example,dc=com
cn: Babs Bayerkohler

dn: uid=user.27553,ou=People,dc=example,dc=com
cn: Babs Bayless

dn: uid=user.14135,ou=People,dc=example,dc=com
cn: Babs Bayley

dn: uid=user.717,ou=People,dc=example,dc=com
cn: Babs Bayly

dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen

dn: uid=user.89830,ou=People,dc=example,dc=com
cn: Babs Pdesupport

dn: uid=user.76412,ou=People,dc=example,dc=com
cn: Babs Peacemaker

dn: uid=user.62994,ou=People,dc=example,dc=com
cn: Babs Peacocke

dn: uid=user.49576,ou=People,dc=example,dc=com
cn: Babs Peake

dn: uid=user.36158,ou=People,dc=example,dc=com
cn: Babs Pearce

dn: uid=user.22740,ou=People,dc=example,dc=com
cn: Babs Pearcy

dn: uid=user.9322,ou=People,dc=example,dc=com
cn: Babs Pearse
```

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 "Schema and JSON" in the *Configuration 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*.

Chapter 4
LDAP Compare

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

In this example, Kirsten Vaughan uses the **ldapcompare** command to check whether the value matches the value of the `description` attribute:

```
$ ldapcompare \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 'description:Description on ou=People' \
 uid=kvaughan,ou=people,dc=example,dc=com
# Comparing type description with value Description on ou=People in entry
 uid=kvaughan,ou=people,dc=example,dc=com
# Compare operation returned true for entry uid=kvaughan,ou=people,dc=example,dc=com
```

Chapter 5
LDAP Updates

Note

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see "Learn About the Evaluation Setup Profile" in the *Installation Guide*.

For details on the LDIF format shown in the examples that follow, see RFC 2849.

Add Entries

- "Add Users"
- "Bulk Adds"

Add Users

The following example adds two new users:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
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
EOF
```

Bulk Adds

The following example adds 10,000 generated entries, using the `--numConnections` option to perform multiple add operations in parallel:

```
# Generate user entries with user IDs larger than those that exist,
# and remove container entries from the output:
$ makeldif \
 --outputLdif output.ldif \
 <(sed "s/<sequential:0>/<sequential:100000>/" /path/to/opendj/config/MakeLDIF/example.template)
$ sed '1,10d' output.ldif > /tmp/generated-users.ldif
# Bulk add the generated user entries:
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --numConnections 64 \
 /tmp/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.

Modify Entries

- "Add Attributes"
- "Change an Attribute"
- "Delete an Attribute"
- "Delete One Attribute Value"
- "From Standard Input"
- "Optimistic Concurrency (MVCC)"
- "JSON Attribute"

Add Attributes

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

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
dn: uid=scarter,ou=people,dc=example,dc=com
changetype: modify
add: description
description: Accounting Manager
-
add: jpegphoto
jpegphoto:<file:///tmp/picture.jpg
EOF
```

Change an Attribute

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

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
dn: uid=scarter,ou=people,dc=example,dc=com
changetype: modify
replace: description
description: New description
EOF
```

Delete an Attribute

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

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
dn: uid=scarter,ou=people,dc=example,dc=com
changetype: modify
delete: jpegphoto
EOF
```

Delete One Attribute Value

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

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
delete: cn
cn: Barbara Jensen
EOF
```

From Standard Input

A double dash, `--`, signifies the end of command options. After the double dash, 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:

```
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 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 -- -
```

```
# Without dashes:
$ cat bjensen-stdin-description.ldif | ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery
```

Optimistic Concurrency (MVCC)

Consider an application that lets end users update user profiles through a browser. It stores user profiles as DS entries. End users can look up user profiles and modify them. The 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.

Suppose two users, Alice and Bob, are 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 simultaneously 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 *LDAP 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 the following. The ETag is a generated value that depends on the content of the entry:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --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 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --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. The application tries the equivalent of the following commands with Bob's updates:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --assertionFilter "(ETag=${ETAG})" << EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: l
l: Grenoble
-
add: description
description: Employee of the Month
EOF
# 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
```

The 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 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --assertionFilter "(ETag=${NEW_ETAG})" << EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: l
l: Grenoble
-
add: description
description: Employee of the Month
EOF
# MODIFY operation successful for DN uid=bjensen,ou=People,dc=example,dc=com
```

JSON Attribute

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

This example depends on the configuration and sample data used in "JSON Query Matching Rule Index" in the *Configuration Guide*. Unless you have installed the server with the evaluation profile, 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:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
dn: uid=bjensen,ou=people,dc=example,dc=com
changetype: modify
replace: json
json: {"stuff":["things","devices","paraphernalia"]}
EOF
```

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 localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --provider-name "Core Schema" \
 --set json-validation-policy:lenient \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Change Incoming Updates

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 possible. 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.

Rename 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 localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --type attribute-cleanup \
 --plugin-name "Rename email to mail" \
 --set enabled:true \
 --set rename-inbound-attributes:email:mail \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Next, confirm that it worked as expected:

```
$ ldapmodify \
 --hostname localhost \
```

```
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
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: chngthspwd
EOF
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=newuser)" \
  mail
dn: uid=newuser,ou=People,dc=example,dc=com
mail: newuser@example.com
```

Remove 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 \
 --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 \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --no-prompt
```

Next, confirm that it worked as expected:

```
$ ldapmodify \
```

```
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
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: chngthspwd
creatorsName: cn=Bad Attr
createTimestamp: Never in a million years.
modifiersName: uid=admin
modifyTimestamp: 20110930164937Z
EOF
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --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>
```

Rename 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`. When you change the RDN of the entry, you rename the entry, modifying the naming attribute and DN.

In this example, 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:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
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
EOF
```

Move Entries

- "Move a Branch"
- "Move an Entry"

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 base DN. Depending on the number of entries you move, this can be a resource-intensive operation.

Move a Branch

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:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: ou=People,dc=example,dc=com
changetype: modrdn
newrdn: ou=Subscribers
deleteoldrdn: 1
newsuperior: dc=example,dc=com
EOF
```

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

Move an Entry

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:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
dn: cn=New App,dc=example,dc=com
changetype: moddn
newrdn: cn=An App
deleteoldrdn: 0
newsuperior: ou=Apps,dc=example,dc=com
EOF
```

Delete Entries

Remove a Branch

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

```
$ dsconfig \
 set-access-control-handler-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --add global-aci:"(targetcontrol=\"SubtreeDelete\")\
 (version 3.0; acl \"Allow Subtree Delete\"; allow(read) \
 userdn=\"ldap:///uid=kvaughan,ou=People,dc=example,dc=com\";)" \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --no-prompt
$ ldapdelete \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
--bindDN "uid=kvaughan,ou=People,dc=example,dc=com" \
--bindPassword bribery \
--deleteSubtree "ou=Special Users,dc=example,dc=com"
```

From Standard Input

A double dash, `--`, signifies the end of command options, After the double dash, 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 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 -- -
```

```
# Without dashes:
$ cat users-to-delete.txt | ldapdelete \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery
```

Chapter 6
LDIF Tools

Generate Test Data

The **makeldif** command uses templates to generate sample data with great flexibility. Default templates are located in the `opendj/config/MakeLDIF/` directory.

Tip

The quickest way to generate user entries is to use the `ds-evaluation` setup profile. The profile lets you generate an arbitrary number of Example.com users as part of the setup process.

For details, see "*Install DS for Evaluation*" in the *Installation Guide*.

1. (Optional) Write a template file for your generated LDIF.

   The `example.template` file used in the examples creates `inetOrgPerson` entries. To learn how to generate test data that matches your production data more closely, read makeldif.template.

2. (Optional) Create additional data files for your template.

   Additional data files are located in the same directory as your template file.

3. (Optional) Decide whether to generate the same test data each time you use the same template.

   If so, provide the same `randomSeed` integer each time you run the command.

4. Run the **makeldif** command to generate your LDIF file.

   The following command demonstrates use of the example MakeLDIF template:

```
$ makeldif \
 --outputLdif generated.ldif \
 --randomSeed 42 \
 /path/to/opendj/config/MakeLDIF/example.template
LDIF processing complete.
```

Search LDIF

The **ldifsearch** command searches for entries in LDIF files:

```
$ ldifsearch \
 --baseDN dc=example,dc=com \
 generated.ldif \
 "(sn=Grenier)" \
 uid
dn: uid=user.4630,ou=People,dc=example,dc=com
uid: user.4630
```

Update LDIF

The **ldifmodify** command applies changes, generating a new version of the LDIF.

In the example that follows, the `changes.ldif` file contains the following LDIF:

```
dn: uid=user.0,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: New description.
-
replace: initials
initials: ZZZ
```

The resulting target LDIF file is approximately the same size as the source LDIF file, but the order of entries in the file is not guaranteed to be identical:

```
$ ldifmodify \
 --outputLdif new.ldif \
 generated.ldif \
 changes.ldif
```

Compare LDIF

The **ldifdiff** command reports differences between two LDIF files in LDIF format:

```
$ ldifdiff generated.ldif new.ldif
dn: uid=user.0,ou=People,dc=example,dc=com
changetype: modify
delete: description
description: This is the description for Aaccf Amar.
-
add: description
description: New description.
-
delete: initials
initials: AAA
-
add: initials
initials: ZZZ
-
```

The **ldifdiff** command reads files into memory to compare their contents. The command is designed to work with small files and fragments, and can quickly run out of memory when calculating the differences between large files.

Use Standard Input

For each LDIF tool, a double dash, `--`, signifies the end of command options. After the double dash, only trailing arguments are allowed.

To indicate standard input as a trailing argument, use a bare dash, `-`, after the double dash. How bare dashes are used after a double dash depends on the tool:

ldifdiff

The bare dash can replace either the source LDIF file, or the target LDIF file argument.

To take the source LDIF from standard input, use the following construction:

```
ldifdiff [options] -- - target.ldif
```

To take the target LDIF from standard input, use the following construction:

```
ldifdiff [options] -- source.ldif -
```

ldifmodify

The bare dash can replace either the *source.ldif* or *changes.ldif* file arguments.

To take the source LDIF from standard input, use the following construction:

```
ldifmodify [options] -- - changes.ldif [changes.ldif ...]
```

To take the changes in LDIF from standard input, use the following construction:

```
ldifmodify [options] -- source.ldif -
```

ldifsearch

The bare dash lets you take the source LDIF from standard input with the following construction:

```
ldifsearch [options] -- - filter [attributes ...]
```

Chapter 7
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 what attributes a given LDAP entry must have and may optionally have, and 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 when you understand how to use LDAP schema.

"*LDAP Schema*" in the *Configuration 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. For the list of standard definitions that DS servers provide, see "Standard Schema" in the *Configuration Guide*.

As a script developer, you use the available schema, and accept the server's application of schema when updating directory entries.

Read Schema

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 "Find LDAP Schema".

Find LDAP Schema

Look up the schema DN:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --baseDN dc=example,dc=com \
 --searchScope base \
 "(&)" \
 subschemaSubentry
dn: dc=example,dc=com
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.

Object Class Schema

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 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --baseDN "cn=schema" \
 --searchScope base \
 "(objectClass=subschema)" \
 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 "Attribute Schema".

Attribute Schema

The following example shows you how to read the schema definition for the `cn` attribute:

```
$ grep \'cn\' <(ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --baseDN "cn=schema" \
 --searchScope base \
 "(objectClass=subschema)" \
 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 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --baseDN "cn=schema" \
 --searchScope base \
 "(objectClass=subschema)" \
 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 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --baseDN "cn=schema" \
 --searchScope base \
 "(objectClass=subschema)" \
 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.

Schema Errors

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 "Update LDAP Schema" in the *Configuration 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. When schema checking is not required, it is sufficient to check 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 is often sufficient 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:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: undefined
undefined: This attribute is not defined.
EOF
# 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 is to define the `undefined` attribute, and to ensure 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:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: organizationalUnit
EOF
# 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:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: cn
cn:
EOF
# 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 "Attribute Schema", a Directory String has one or more UTF-8 characters.

Workarounds

Follow the suggestions in "Schema Errors" 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.

ExtensibleObject

The following example adds one attribute that is undefined and another that is not allowed:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery << EOF
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.
EOF
# MODIFY operation successful for DN uid=bjensen,ou=People,dc=example,dc=com
```

Use of the `extensibleObject` object class can be abused and can prevent interoperability. Restrict its use to cases where no better alternative is available.

Chapter 8
Passwords

Note

Examples in this documentation depend on features activated in the `ds-evaluation` setup profile.

For details, see "Learn About the Evaluation Setup Profile" in the *Installation Guide*.

- "Reset a Password"
- "Change Your Password"
- "Check Password Quality"
- "Passwords With Special Characters"

The **ldappasswordmodify** command lets authorized users change their own passwords and reset other users' passwords.

Reset a Password

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.

Assume password administrator Kirsten Vaughan has the `password-reset` privilege. The following example shows Kirsten resetting Andy Hall's password:

```
$ ldappasswordmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --authzID "dn:uid=ahall,ou=people,dc=example,dc=com"
The LDAP password modify operation was successful
Generated Password:  <password>
```

+ *More Information*

If a client application performs the LDAP password modify extended operation on a connection that is bound to 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.

To change, rather than reset, the password as the user while binding as an application or an administrator, use the LDAP Password Modify extended operation with an authorization ID. Alternatively, use proxied authorization, as described in "*Proxied Authorization*".

If you reset a password, and do not want it to count as a password reset, use the **manage-account** command with the **set-password-is-reset** hidden option, supported only for testing:

```
$ manage-account \
 set-password-is-reset \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword password \
 --targetDN uid=ahall,ou=people,dc=example,dc=com \
 --operationValue true \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin
```

Change Your Password

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

```
$ ldappasswordmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "uid=ahunter,ou=people,dc=example,dc=com" \
 --bindPassword egregious \
 --newPassword chngthspwd
```

The same operation works for directory superusers, such as `uid=admin`:

```
$ ldappasswordmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --authzID dn:uid=admin \
 --currentPassword password \
 --newPassword OzNOkkfkTJDSW9Bg
```

Check Password Quality

The **ldappasswordmodify** and **ldapmodify** commands support password quality advice controls to get additional information about why a password update failed. When you use the request control and a password update fails, the server can send the response control with details indicating which validators rejected the new password.

Note

The new LDAP control has interface stability: Evolving.

The following commands demonstrate how the tools show the information from the response control:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetcontrol="PasswordQualityAdvice") (version 3.0; acl
  "Authenticated users can check password quality";
  allow(read) userdn="ldap:///all";)
EOF
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: cn=Minimum length policy,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: ds-pwp-password-policy
objectClass: ds-pwp-validator
objectClass: ds-pwp-length-based-validator
cn: Minimum length policy
ds-pwp-password-attribute: userPassword
ds-pwp-default-password-storage-scheme: PBKDF2-HMAC-SHA512
ds-pwp-length-based-min-password-length: 8
subtreeSpecification: {base "ou=people", specificationFilter "(uid=pshelton)" }
EOF
$ ldappasswordmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
```

```
 --bindDN uid=pshelton,ou=People,dc=example,dc=com \
 --bindPassword nosedive \
 --control PasswordQualityAdvice:true \
 --control NoOp \
 --newPassword passwd
The LDAP password modify operation failed: 19 (Constraint Violation)
Additional Information:  The provided new password failed the validation
checks defined in the server: The provided password is shorter than the
minimum required length of 8 characters

The new password was rejected by the password policy located in "cn=Minimum
length policy,dc=example,dc=com"

The following password quality criteria were not satisfied:
* length-based with parameters {max-password-length=0, min-password-length=8}
```

Notice that the check can be performed as a no-op.

Passwords With Special Characters

DS servers expect passwords to be UTF-8 encoded and base64-encoded when included in LDIF. UTF-8 characters such as `à` or `ô` must be correctly encoded:

```
$ export LANG=en_US.UTF-8
$ ldappasswordmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=wlutz,ou=People,dc=example,dc=com \
 --bindPassword bassinet \
 --newPassword pàsswôrd
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --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
```

Chapter 9

Proxied Authorization

Proxied authorization, defined in RFC 4370, provides a mechanism for binding as a proxy, and making requests on behalf of other users. For example, an application binds with its credentials, but each request is made as a user who logs in through the application.

To use proxied authorization, the proxy user must have:

- Permission to use the LDAP Proxy Authorization Control.

  Grant access to this control using an ACI with a `targetcontrol` list that includes the Proxy Authorization Control OID `ProxiedAuthV2` (`2.16.840.1.113730.3.4.18`). The ACI must grant `allow(read)` permission to the proxy.

  This calls for an ACI with a target scope that includes the entry of the proxy 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. The ACI must grant `allow(proxy)` permission to the proxy.

- The privilege to use proxied authorization.

  Add `ds-privilege-name: proxied-auth` to the proxy's entry.

The following table shows whether proxied authorization allows an operation on the target.

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

Note

Resource limits do not change when the user proxies as another user. Resource limits depend on the bind DN, not the proxy authorization identity.

This following steps demonstrate proxied authorization for an Example.com application:

1. Grant access to applications to use the Proxy Authorization control, and to use proxied authorization:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetcontrol="ProxiedAuthV2")
  (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");)
EOF
```

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 Example.com user, but cannot proxy as the directory superuser `uid=admin`. The target of the ACI does not include `uid=admin`.

2. Grant My App the privilege to use proxied authorization:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password << EOF
dn: cn=My App,ou=Apps,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: proxied-auth
EOF
```

Other applications without this privilege cannot yet use proxied authorization.

3. Test that My App can use proxied authorization:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN "cn=My App,ou=Apps,dc=example,dc=com" \
 --bindPassword password \
 --proxyAs "dn:uid=kvaughan,ou=People,dc=example,dc=com" << EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Changed through proxied auth
EOF
# MODIFY operation successful for DN uid=bjensen,ou=People,dc=example,dc=com
```

Use an identity mapper if identifiers have the `u:authzid` (user ID) form rather than `dn:authzid` form. Specify the identity mapper with the global configuration setting, `proxied-authorization-identity-mapper` in the *Configuration Reference*.

For details, see "Identity Mappers".

Chapter 10
Notification of Changes

Applications that need change notification can use a persistent search or read the external change log:

- "Use Persistent Search"
- "Use the External Change Log"

Use Persistent Search

Defined in the Internet-Draft, *Persistent Search: A Simple LDAP Change Notification Mechanism*, a persistent search is like a regular search that never returns. Every time a change happens in the scope of the search, the server returns an additional response:

1. Grant access to perform a persistent search, by adding an ACI to use the persistent search control.

   Persistent searches consume server resources, so servers do not allow them by default. If an application does not have access, the request fails with an unavailable critical extension error:

   ```
   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
   ```

   The following command grants access under `dc=example,dc=com` to `My App`:

   ```
   $ ldapmodify \
    --hostname localhost \
    --port 1636 \
    --useSsl \
    --usePkcs12TrustStore /path/to/opendj/config/keystore \
    --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
    --bindDN uid=admin \
    --bindPassword password << EOF
   dn: dc=example,dc=com
   changetype: modify
   add: aci
   aci: (targetcontrol = "PSearch")
    (version 3.0;acl "Allow Persistent Search for My App";
    allow (read)(userdn = "ldap:///cn=My App,ou=Apps,dc=example,dc=com");)
   EOF
   ```

2. Start the persistent search.

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

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN 'cn=My App,ou=Apps,dc=example,dc=com' \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 --persistentSearch ps:all:true:false \
 '(&)' >> /tmp/psearch.txt &
$ export PSEARCH_PID=$!
```

Notice the search filter, `(&)`, which is always true, meaning that it matches all entries. For details on settings for a persistent search, see the `--persistentSearch` option in "Options" in the *Tools Reference*.

3. Make changes that impact the persistent search results.

   + *Show commands*

   To prepare to modify an entry, download the LDIF changes :

   ```
   dn: uid=bjensen,ou=People,dc=example,dc=com
   changetype: modify
   replace: description
   description: Hello, persistent search
   ```

   The following commands perform a modify operation and a delete operation:

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery << EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Hello, persistent search
EOF
$ ldapdelete \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=kvaughan,ou=People,dc=example,dc=com \
 --bindPassword bribery \
 uid=tpierce,ou=People,dc=example,dc=com
```

+ *Show persistent search results*

The result is 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: {PBKDF2-HMAC-SHA256}10000:<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: {PBKDF2-HMAC-SHA256}10000:<hash>
uidNumber: 1042
uid: tpierce
l: Bristol
departmentNumber: 1000
preferredLanguage: en-gb
street: Broad Quay House, Prince Street
```

If the data is replicated, the results include the entry `dc=example,dc=com`. Replication updates the `ds-sync-*` operational attributes on `dc=example,dc=com`, and those changes appear in the results because the entry is in the scope of the persistent search.

4. Terminate the persistent search.

   Interrupt the command with **CTRL+C** (`SIGINT`) or `SIGTERM`:

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

Use the External Change Log

You read the external change log over LDAP. When you poll the change log, you can get the list of updates that happened since your last request.

The external change log mechanism uses an LDAP control with OID `1.3.6.1.4.1.26027.1.5.4`. This control allows the client application to bookmark the last changes seen. The control returns a cookie that the application sends to the server to read the next batch of changes.

These steps show the client binding as `uid=admin` to read the change log. Other accounts require sufficient access and privileges to read the change log. For instructions, see "Let a User Read the Changelog" in the *Configuration Guide*:

1. Send an initial search request using the LDAP control with no cookie value.

   In this example, two changes appear in the changelog:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN cn=changelog \
 --control "ecl:false" \
 "(&)" \
 changes changeLogCookie targetDN
# Public changelog exchange control(1.3.6.1.4.1.26027.1.5.4): <COOKIE1>
dn: replicationCSN=<CSN1>,dc=example,dc=com,cn=changelog
changes::
 cmVwbGFjZTogZGVzY3JpcHRpb24KZGVzY3JpcHRpb246IE5ldyBkZXNjcmlwdGlvbgotCnJlcGxhY2U6IG1vZGlmaWVyc05hbWUKbW9kaWZpZX
targetDN: uid=bjensen,ou=People,dc=example,dc=com
changeLogCookie: <COOKIE1>

# Public changelog exchange control(1.3.6.1.4.1.26027.1.5.4): <COOKIE2>
dn: replicationCSN=<CSN2>,dc=example,dc=com,cn=changelog
changes::
 cmVwbGFjZTogZGVzY3JpcHRpb24KZGVzY3JpcHRpb246IE5ldywgaW1wcm92ZWQgZGVzY3JpcHRpb24KLQpyZXBsYWNlOiBtb2RpZmllcnNOYW
targetDN: uid=bjensen,ou=People,dc=example,dc=com
changeLogCookie: <COOKIE2>
```

The changes are base64-encoded. You can decode them using the **base64** command. The following example decodes the first change:

```
$ base64 decode --encodedData
 cmVwbGFjZTogZGVzY3JpcHRpb24KZGVzY3JpcHRpb246IE5ldyBkZXNjcmlwdGlvbgotCnJlcGxhY2U6IG1vZGlmaWVyc05hbWUKbW9kaWZpZX
replace: description
description: New description
-
replace: modifiersName
modifiersName: uid=bjensen,ou=People,dc=example,dc=com
-
replace: modifyTimestamp
modifyTimestamp: <timestamp>
-
```

Notice the `changeLogCookie` value, which has the form *`base-dn:CSN`*. Here, *CSN* is a change sequence number in the *Getting Started*.

2. To start reading a particular change in the changelog, provide the cookie with the control:

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN cn=changelog \
 --control "ecl:false:$COOKIE1" \
 "(&)" \
 changes changeLogCookie targetDN
# Public changelog exchange control(1.3.6.1.4.1.26027.1.5.4): <COOKIE2>
dn: replicationCSN=<CSN2>,dc=example,dc=com,cn=changelog
changes::
 cmVwbGFjZTogZGVzY3JpcHRpb24KZGVzY3JpcHRpb246IE5ldywgaW1wcm92ZWQgZGVzY3JpcHRpb24KLQpyZXBsYWNlOiBtb2RpZmllcnNOYW
targetDN: uid=bjensen,ou=People,dc=example,dc=com
changeLogCookie: <COOKIE2>
```

The following command decodes the changes:

```
$ base64 decode --encodedData
 cmVwbGFjZTogZGVzY3JpcHRpb24KZGVzY3JpcHRpb246IE5ldywgaW1wcm92ZWQgZGVzY3JpcHRpb24KLQpyZXBsYWNlOiBtb2RpZmllcnNOYW
replace: description
description: New, improved description
-
replace: modifiersName
modifiersName: uid=bjensen,ou=People,dc=example,dc=com
-
replace: modifyTimestamp
modifyTimestamp: <timestamp>
-
```

3. If you lose the cookie, start over from the earliest available change by sending a request with no cookie.