Start here

Use this guide to get a quick, hands-on look at what Directory Services software can do. You will download, install, and use DS on your local computer.

Expect to spend 30-120 minutes working through this guide.

Install DS

Install DS software.

Learn LDAP

Use DS LDAP tools.

Learn REST/HTTP

Access DS over HTTP.

Learn Replication

Replicate DS data.

Measure Performance

Measure LDAP operations.

Learn Access Control

Learn DS ACIs.

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

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

Install DS

TIP

DS software has no GUI. Instead, DS software is bundled with command-line tools.

Because LDAP is standard, you can use third-party GUI tools to view and edit directory data. For a short list, see Try Third-Party Tools.

Prepare For Installation

1. To evaluate DS software, make sure you have 10 GB free disk space for the software and for sample data.
2. Verify that you have a supported Java version installed on your local computer.

   For details, check the supported Java versions.
3. If you plan to run the Bash-based REST API examples, make sure the **`curl`** command is available.

   For details, see the curl site.

Download DS Software

Directory Services software is free to download, evaluate, and use in development:

1. If you do not have an account on ForgeRock BackStage, sign up for one.
2. Sign in to ForgeRock BackStage.
3. Find and download the latest Directory Services ZIP distribution.

Install a Directory Server

1. Unzip the `.zip` file into the file system directory where you want to install the server.

   Unzipping the .zip file creates a top-level `opendj` directory in the directory where you unzipped the file. On Windows systems if you unzip the file with Right-Click > Extract All, remove the trailing `opendj-`*`version`* directory from the folder you specify.

The documentation shows the installation file system directory as `/path/to/opendj`.

For example:

1. Bash
2. PowerShell
3. Zsh

```
$ unzip ~/Downloads/DS-7.1.8.zip -d /path/to
```

```
PS C:\path\to> Expand-Archive DS-7.1.8.zip C:\path\to
```

```
% unzip ~/Downloads/DS-7.1.8.zip -d /path/to
```

2. Use the **`setup`** command to set up a server with the `ds-evaluation` profile. The evaluation profile includes Example.com sample data, more lenient access control, and some other features.

   The following example runs the command non-interactively. Use the same settings shown here to be able to copy and paste the commands shown in this guide:

   1. Bash
   2. PowerShell
   3. Zsh

```
$ /path/to/opendj/setup \
 --serverId first-ds \
 --deploymentKeyPassword password \
 --rootUserDn uid=admin \
 --rootUserPassword password \
 --monitorUserPassword password \
 --hostname localhost \
 --ldapPort 1389 \
 --ldapsPort 1636 \
 --httpsPort 8443 \
 --adminConnectorPort 4444 \
 --replicationPort 8989 \
 --profile ds-evaluation \
 --start \
 --acceptLicense

Validating parameters..... Done
```

```
Configuring certificates..... Done

Store the following deployment key in a safe place and re-
use it when
configuring other servers in the topology:

<deployment-key>

Configuring server... Done
Configuring profile DS evaluation.....................
Done
Starting directory server............... Done

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

```
PS C:\path\to> C:\path\to\opendj\setup.bat `
 --serverId first-ds `
 --deploymentKeyPassword password `
 --rootUserDn uid=admin `
 --rootUserPassword password `
 --monitorUserPassword password `
 --hostname localhost `
 --ldapPort 1389 `
 --ldapsPort 1636 `
 --httpsPort 8443 `
 --adminConnectorPort 4444 `
 --replicationPort 8989 `
 --profile ds-evaluation `
 --start `
 --acceptLicense
```

```
% /path/to/opendj/setup \
 --serverId first-ds \
 --deploymentKeyPassword password \
 --rootUserDn uid=admin \
 --rootUserPassword password \
 --monitorUserPassword password \
 --hostname localhost \
 --ldapPort 1389 \
 --ldapsPort 1636 \
 --httpsPort 8443 \
 --adminConnectorPort 4444 \
 --replicationPort 8989 \
 --profile ds-evaluation \
 --start \
 --acceptLicense
```

Save the generated `<deployment-key>`. You will use this key later when setting up a second server for replication.

▼ More about setup options

The **`setup`** command shown here has the following options:

`--rootUserDn uid=admin`

These options set the credentials for the directory superuser. This user has privileges to perform any and all administrative operations, and is not subject to access control. It is called the *root user* due to the similarity to the UNIX root user.

The root user *distinguished name* (DN) identifies the directory superuser. In LDAP, a DN is the fully qualified name for a directory entry. The name used here is the default name: `uid=admin`.

`--monitorUserPassword password`

The monitor user has the privilege to read monitoring data. No `--monitorUserDn` option is set, so the DN defaults to `uid=Monitor`.

`--hostname localhost`

The server uses the fully qualified domain name for identification between replicated servers.

Using `localhost` is a shortcut suitable only for evaluation on your local computer. In production, set this to the fully qualified domain name, such as `ds.example.com`.

`--ldapPort 1389`

The reserved port for LDAP is `389`. Connections to this port can be secured with StartTLS, but are not secure by default.

Examples in the documentation use `1389`, which is accessible to non-privileged users.

`--ldapsPort 1636`

The reserved port for LDAPS is `636`. Connections to this port are secured with TLS.

Examples in the documentation use `1636`, which is accessible to non-privileged users.

`--httpsPort 8443`

The reserved port for HTTPS is `443`.

HTTP client applications access directory data and monitoring information on this port.

Examples in the documentation use `8443`, which is accessible to non-privileged users.

`--adminConnectorPort 4444`

This is the service port used to configure the server and to run tasks. Connections to this port are secured with TLS.

The port used in the documentation is `4444`, which is the initial port suggested during interactive setup.

`--replicationPort 8989`

This is the service port used for replication messages.

The port used in the documentation is `8989`, which is the initial port suggested during interactive setup.

`--profile ds-evaluation`

The setup profile adds hard-coded entries for users like Babs Jensen, and groups like Directory Administrators. It also generates 100,000 sample LDAP user entries. All generated users have the same password, literally `password`. The generated user accounts are helpful for performance testing.

All entries are added under the base DN `dc=example,dc=com`. A base DN is the suffix shared by all DNs in a set of directory data.

LDAP entries are arranged hierarchically in the directory. The hierarchical organization resembles a file system on a PC or a web server, often visualized as an upside down tree structure, or a pyramid. In the same way a full path uniquely identifies each file or folder in a file system, a DN uniquely identifies each LDAP entry.

Each DN consists of components separated by commas, such as `uid=bjensen,ou=People,dc=example,dc=com`. The base DN matches the final components of each DN in that branch of the directory. A DN's components reflect the hierarchy of directory entries. The user entry with DN `uid=bjensen,ou=People,dc=example,dc=com` is under the organizational unit entry `ou=People,dc=example,dc=com`, which in turn is under `dc=example,dc=com`.

Basic components have the form `*attribute-name*=*attribute-value*`, such as `dc=com`. In the example `dc=com`, the attribute `dc` (DNS domain component) has the value `com`. The DN `dc=example,dc=com` reflects the DNs domain name `example.com`.

--start

By default, the **`setup`** command does not start the server. This lets you complete any necessary configuration steps before starting the server for the first time, which may initiate the replication process.

In this case, you have no further configuration to do. This option causes the server to start immediately.

--acceptLicense

Remove this option to read the license and then accept it interactively.

Alternatively, you can run the **`setup`** command interactively by starting it without options.

3. Add the DS tools to your PATH to avoid having to specify the full path for each command:

   1. Bash
   2. PowerShell
   3. Zsh

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

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

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

4. Run the **`status`** command:

   1. Bash
   2. PowerShell
   3. Zsh

```
$ status \
 --bindDn uid=admin \
 --bindPassword password \
 --hostname localhost \
 --port 4444 \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin
```

```
PS C:\path\to> status.bat `
 --bindDn uid=admin `
 --bindPassword password `
 --hostname localhost `
 --port 4444 `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file
C:\path\to\opendj\config\keystore.pin
```

```
% status \
 --bindDn uid=admin \
 --bindPassword password \
 --hostname localhost \
 --port 4444 \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file
/path/to/opendj/config/keystore.pin
```

The `status` command uses a secure connection to the administration port. To trust the server's certificate, the command uses the server's own truststore.

Read the output that the `status` command displays.

Learn LDAP

LDAP is short for Lightweight Directory Access Protocol, a standard Internet protocol. The examples that follow show you how to use bundled DS command-line tools to send LDAP requests.

Before you try the examples, set up a server, as described in Install DS. Make sure you added the command-line tools to your PATH:

1. Bash
2. PowerShell
3. Zsh

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

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

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

Search

Searching the directory is like searching for a phone number in a phone book. You can look up a subscriber's phone number because you know the subscriber's last name. In other words, you use the value of an attribute to find entries that have attributes of interest.

When looking up a subscriber's entry in a phone book, you need to have some idea where they live in order to pick the right phone book. For example, a Los Angeles subscriber cannot be found in the New York phone book. In an LDAP directory, you need to know at least the base DN to search under.

For this example, assume you know a user's full name, `Babs Jensen`, and that Babs Jensen's entry is under the base DN `dc=example,dc=com`. You want to look up Babs Jensen's email and office location. The following command sends an appropriate LDAP search request to the server you installed:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDn dc=example,dc=com \
 "(cn=Babs Jensen)" \
 cn mail street l

dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
l: San Francisco
mail: bjensen@example.com
street: 201 Mission Street Suite 2900
```

```
PS C:\path\to> ldapsearch.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --baseDn dc=example,dc=com `
 "(cn=Babs Jensen)" `
 cn mail street l

dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
l: San Francisco
mail: bjensen@example.com
street: 201 Mission Street Suite 2900
```

```
% ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDn dc=example,dc=com \
 "(cn=Babs Jensen)" \
 cn mail street l

dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
l: San Francisco
mail: bjensen@example.com
street: 201 Mission Street Suite 2900
```

▼ More about the search example

Notice the following characteristics of the search:

- The command makes a secure connection to the server using LDAPS.

  The command relies on the server’s truststore to trust the CA certificate used to sign the server certificate.

- The base DN option, `--baseDn dc=example,dc=com`, tells the server where to look for Babs Jensen’s entry. Servers can hold data for multiple base DNs, so this is important information.

  It is possible to restrict the scope of the search, but the default is to search the entire subtree under the base DN.

- The command uses a *search filter*, `"(cn=Babs Jensen)"`, which tells the server, "Find entries whose `cn` attribute exactly matches the string `Babs Jensen` without regard to case."

  The `cn` (`commonName`) attribute is a standard attribute for full names.

  Internally, the directory server has an equality index for the `cn` attribute. The directory uses the index to quickly find matches for `babs jensen`. The default behavior in LDAP is to ignore case, so `"(cn=Babs Jensen)"`, `"(cn=babs jensen)"`, and `"(CN=BABS JENSEN)"` are equivalent.

  If more than one entry matches the filter, the server returns multiple entries.

- The filter is followed by a list of LDAP attributes, `cn mail street l`. This tells the server to return only the specified attributes in the search result entries. By default, if you do not specify the attributes to return, the server returns all the user attributes that you have the right to read.

- The result shows attributes from a single entry. Notice that an LDAP entry, represented here in the standard LDIF format, has a flat structure with no nesting.

  The DN that uniquely identifies the entry is `uid=bjensen,ou=People,dc=example,dc=com`. Multiple entries can have the same attribute values, but each must have a unique DN. This is the same as saying that the leading *relative distinguished name* (RDN) value must be unique at this level in the hierarchy. Only one entry directly under `ou=People,dc=example,dc=com` has the RDN `uid=bjensen`.

  The `mail`, `street`, `l` (location), and `uid` attributes are all standard LDAP attributes like `cn`.

For additional examples, see LDAP Search.

Modify

You installed the server with the `ds-evaluation` profile. That profile grants access to search Example.com data without authenticating to the directory. When modifying directory data, however, you must authenticate first. LDAP servers must know who you are to determine what you have access to.

In the following example Babs Jensen modifies the description on her own entry:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=bjensen,ou=People,dc=example,dc=com \
 --bindPassword hifalutin <<EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: New description
EOF

# MODIFY operation successful for DN
uid=bjensen,ou=People,dc=example,dc=com
```

```
PS C:\path\to> New-Item -Path . -Name "description.ldif" -ItemType
"file" -Value @"
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: New description
"@
PS C:\path\to> ldapmodify.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn uid=bjensen,ou=People,dc=example,dc=com `
 --bindPassword hifalutin `
 description.ldif

# MODIFY operation successful for DN
uid=bjensen,ou=People,dc=example,dc=com
```

```
% ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=bjensen,ou=People,dc=example,dc=com \
 --bindPassword hifalutin <<EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: New description
EOF

# MODIFY operation successful for DN
uid=bjensen,ou=People,dc=example,dc=com
```

▼ More about the modify example

- Babs Jensen's authentication credentials are provided with the `--bindDn` and `--bindPassword` options. Notice that the user identifier is Babs Jensen's DN.

  Authentication operations bind an LDAP identity to a connection. In LDAP, a client application connects to the server, then binds an identity to the connection. An LDAP client application keeps its connection open until it finishes performing its operations. The server uses the identity bound to the connection to make authorization decisions for subsequent operations, such as search and modify requests.

  If no credentials are provided, then the identity for the connection is that of an anonymous user. As a directory administrator, you can configure access controls for anonymous users just as you configure access controls for other users.

  A simple bind involving a DN and a password is just one of several supported authentication mechanisms. The documentation frequently shows simple binds in examples because this kind of authentication is so familiar. Alternatives include authenticating with a digital certificate, or using Kerberos.

- The modification is expressed in standard LDAP Data Interchange Format (LDIF).

  The LDIF specifies the DN of the target entry to modify. It then indicates that the change to perform is an LDAP modify, and that the value `New description` is to replace existing values of the `description` attribute.

- Notice that the result is a comment indicating success. The command's return code—0, but not shown in the example—also indicates success.

  The scripts and applications that you write should use and trust LDAP return codes.

For additional examples, see LDAP Updates, and Passwords.

Add

Authorized users can modify attributes, and can also add and delete directory entries.

The following example adds a new user entry to the directory:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=admin \
 --bindPassword password <<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
mail: newuser@example.com
userPassword: chngthspwd
EOF

# ADD operation successful for DN
uid=newuser,ou=People,dc=example,dc=com
```

```
PS C:\path\to> New-Item -Path . -Name "user.ldif" -ItemType "file"
-Value @"
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
mail: newuser@example.com
```

```
userPassword: chngthspwd
"@
PS C:\path\to> ldapmodify.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn uid=admin `
 --bindPassword password `
 user.ldif

# ADD operation successful for DN
uid=newuser,ou=People,dc=example,dc=com
```

```
% ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=admin \
 --bindPassword password <<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
mail: newuser@example.com
userPassword: chngthspwd
EOF

# ADD operation successful for DN
uid=newuser,ou=People,dc=example,dc=com
```

▼ More about the add example

- The bind DN for the user requesting the add is `uid=admin`. It is also possible to authorize regular users to add entries.
- The entry to add is expressed in standard LDIF.

For additional examples, see LDAP Updates.

Delete

The following example deletes the user added in Add:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapdelete \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=admin \
 --bindPassword password \
 uid=newuser,ou=People,dc=example,dc=com

# DELETE operation successful for DN
uid=newuser,ou=People,dc=example,dc=com
```

```
PS C:\path\to> ldapdelete.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn uid=admin `
 --bindPassword password `
 uid=newuser,ou=People,dc=example,dc=com

# DELETE operation successful for DN
uid=newuser,ou=People,dc=example,dc=com
```

```
% ldapdelete \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=admin \
 --bindPassword password \
 uid=newuser,ou=People,dc=example,dc=com

# DELETE operation successful for DN
uid=newuser,ou=People,dc=example,dc=com
```

Notice that the **`ldapdelete`** command specifies the entry to delete by its DN.

For additional examples, see LDAP Updates.

Learn REST/HTTP

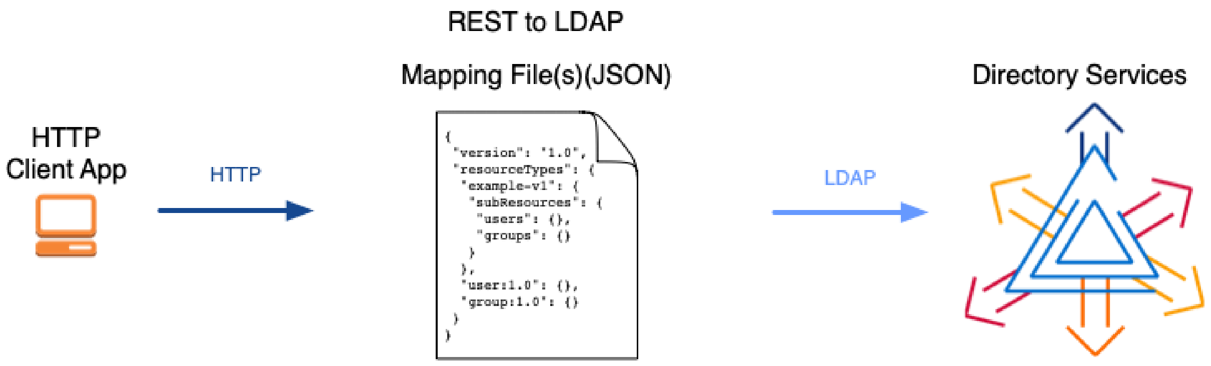
The examples that follow show you how to send RESTful HTTP requests to the directory server.

Before you try the examples, set up a server, as described in Install DS.

▼ More about HTTP access

ForgeRock Directory Services let you access directory data over HTTP as well as LDAP. The feature is known as REST to LDAP because it transforms REST operations into LDAP operations.

The server maps JSON resources to LDAP entries in order to convert HTTP operations to LDAP internally. The directory server you installed is bundled with a default mapping file for sample data. You can configure your own mapping depending on your directory data, and the JSON objects you want.



```
{
 "version": "1.0",
 "resourceTypes": {
  "example-v1": {
   "subResources": {
    "users": {},
    "groups": {}
   }
  },
  "user:1.0": {},
  "group:1.0": {}
 }
}
```

Prepare

Get the deployment CA certificate used to trust the server in the RESTful examples:

▼ (Bash, Zsh) Get the CA cert in PEM format

When using the Bash or Zsh examples with **`curl`** commands, get the deployment CA certificate in PEM format, which the **`curl`** command can read:

1. Bash
2. Zsh

```
$ dskeymgr \
 export-ca-cert \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --outputFile ca-cert.pem
```

```
% dskeymgr \
 export-ca-cert \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --outputFile ca-cert.pem
```

▼ (PowerShell) Import the CA certificate with MMC

For PowerShell examples, first configure Windows to trust the deployment CA certificate. Import the deployment CA from the server truststore using Microsoft Management Console (MMC):

1. Run Microsoft Management Console (**`mmc.exe`**).
2. Add the certificates snap-in so you can import the deployment CA certificate:
   - In the console, select File > Add/Remove Snap-in, then Add.
   - Select Certificates from the list of snap-ins and click Add.
   - Finish the wizard.
3. Import the deployment CA certificate using the snap-in:
   - Select Console Root > Trusted Root Certification Authorities > Certificates.
   - In the Action menu, select Import to open the wizard.
   - Use the wizard to import the deployment CA certificate from the server truststore file, `C:\path\to\opendj\config\keystore` .

     The truststore password is the text in the file `C:\path\to\opendj\config\keystore.pin` .

Create

Use the REST API to create a user resource over HTTP:

1. Bash
2. PowerShell
3. Zsh

```
$ curl \
 --request POST \
 --cacert ca-cert.pem \
 --user admin:password \
 --header "Content-Type: application/json" \
 --data '{
  "_id": "newuser",
  "_schema":"frapi:opendj:rest2ldap:user:1.0",
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1212",
    "emailAddress": "newuser@example.com"
  },
  "name": {
    "givenName": "New",
    "familyName": "User"
  },
  "displayName": ["New User"],
  "manager": {
    "_id": "bjensen",
    "displayName": "Babs Jensen"
  }
 }' \
 https://localhost:8443/api/users?_prettyPrint=true

{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<timestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName": "New",
    "familyName": "User"
  },
  "manager" : {
    "_id" : "bjensen",
    "_rev" : "<revision>"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  }
}
```

```
PS C:\path\to> $Credentials =
[System.Convert]::ToBase64String([System.Text.Encoding]::ASCII.Get
Bytes("admin:password"))
$Headers = @{
    Authorization = "Basic $Credentials"
}
Invoke-RestMethod `
 -Uri https://localhost:8443/api/users `
 -Method Post `
 -Headers $Headers `
 -ContentType application/json `
 -Body @"
 {
  "_id": "newuser",
  "_schema":"frapi:opendj:rest2ldap:user:1.0",
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1212",
    "emailAddress": "newuser@example.com"
  },
  "name": {
    "givenName": "New",
    "familyName": "User"
  },
  "displayName": ["New User"],
  "manager": {
    "_id": "bjensen",
    "displayName": "Babs Jensen"
  }
 }
"@ | ConvertTo-JSON

{
    "_id":  "newuser",
    "_rev":  "<revision>",
    "_schema":  "frapi:opendj:rest2ldap:user:1.0",
    "_meta":  {
                  "created":  "<timestamp>"
```

```
              },
    "userName":  "newuser@example.com",
    "displayName":  [
                        "New User"
                    ],
    "name":  {
                 "givenName":  "New",
                 "familyName":  "User"
             },
    "manager":  {
                    "_id":  "bjensen",
                    "_rev":  "<revision>"
                },
    "contactInformation":  {
                               "telephoneNumber":  "+1 408 555
1212",
                               "emailAddress":
"newuser@example.com"
                           }
}
```

```
% curl \
 --request POST \
 --cacert ca-cert.pem \
 --user admin:password \
 --header "Content-Type: application/json" \
 --data '{
  "_id": "newuser",
  "_schema":"frapi:opendj:rest2ldap:user:1.0",
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1212",
    "emailAddress": "newuser@example.com"
  },
  "name": {
    "givenName": "New",
    "familyName": "User"
  },
  "displayName": ["New User"],
  "manager": {
    "_id": "bjensen",
    "displayName": "Babs Jensen"
  }
 }' \
 "https://localhost:8443/api/users?_prettyPrint=true"
```

```
{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<timestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName": "New",
    "familyName": "User"
  },
  "manager" : {
    "_id" : "bjensen",
    "_rev" : "<revision>"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  }
}
```

▼ More about the example

- The command makes a secure connection to the server using HTTPS.
- The user performing the HTTP POST is the directory superuser.

  The default authorization mechanism for HTTP access is HTTP Basic authentication. The superuser's HTTP user ID, `admin`, is mapped to the LDAP DN, `uid=admin`. REST to LDAP uses the DN and password to perform a simple LDAP bind for authentication. The directory can then use its LDAP-based access control mechanisms to authorize the operation.
- The form of the JSON data respects the API defined by the mapping file. The example mapping file is a JSON format configuration file with comments. See `/path/to/opendj/config/rest2ldap/endpoints/api/example-v1.json` .
- The successful response is the JSON resource that the command created.

  Fields whose names start with `_` are reserved. For details, see DS REST APIs.

For additional details, see DS REST APIs and Create.

Read

Use the REST API to read the user resource created in Create:

1. Bash
2. PowerShell
3. Zsh

```
$ curl \
 --request GET \
 --cacert ca-cert.pem \
 --user bjensen:hifalutin \
 --header "Content-Type: application/json" \
 https://localhost:8443/api/users/newuser?_prettyPrint=true

{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<timestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName": "New",
    "familyName": "User"
  },
  "manager" : {
    "_id" : "bjensen",
    "_rev" : "<revision>"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  }
}
```

```
PS C:\path\to> $Credentials =
[System.Convert]::ToBase64String([System.Text.Encoding]::ASCII.Get
Bytes("bjensen:hifalutin"))
$Headers = @{
    Authorization = "Basic $Credentials"
}
Invoke-RestMethod `
 -Uri https://localhost:8443/api/users/newuser `
 -Method Get `
 -Headers $Headers `
 -ContentType application/json | ConvertTo-JSON

{
    "_id":  "newuser",
    "_rev":  "<revision>",
    "_schema":  "frapi:opendj:rest2ldap:user:1.0",
    "_meta":  {
                  "created":  "<timestamp>"
              },
    "userName":  "newuser@example.com",
    "displayName":  [
                        "New User"
                    ],
    "name":  {
                 "givenName":  "New",
                 "familyName":  "User"
             },
    "manager":  {
                    "_id":  "bjensen",
                    "_rev":  "<revision>"
                },
    "contactInformation":  {
                               "telephoneNumber":  "+1 408 555
1212",
                               "emailAddress":
"newuser@example.com"
                           }
}
```

```
% curl \
 --request GET \
 --cacert ca-cert.pem \
 --user bjensen:hifalutin \
 --header "Content-Type: application/json" \
 "https://localhost:8443/api/users/newuser?_prettyPrint=true"

{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<timestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "New User" ],
  "name" : {
    "givenName": "New",
    "familyName": "User"
  },
  "manager" : {
    "_id" : "bjensen",
    "_rev" : "<revision>"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  }
}
```

Notice that Babs Jensen authenticates for this HTTP GET request. If no credentials are specified, the response is the HTTP 401 Unauthorized:

```
{"code":401,"reason":"Unauthorized","message":"Invalid
Credentials"}
```

In other words, the HTTP Basic authorization mechanism requires authentication even for read operations.

For additional details, see DS REST APIs and Read. You can also query collections of resources, as described in Query.

Update

Use the REST API to replace the user resource created in Create:

1. Bash
2. PowerShell
3. Zsh

```
$ curl \
 --request PUT \
 --cacert ca-cert.pem \
 --user admin:password \
 --header "Content-Type: application/json" \
 --header "If-Match: *" \
 --data '{
  "_id": "newuser",
  "_schema":"frapi:opendj:rest2ldap:user:1.0",
  "contactInformation": {
    "telephoneNumber": "+1 234 567 8910",
    "emailAddress": "updated.user@example.com"
  },
  "name": {
    "givenName": "Updated",
    "familyName": "User"
  },
  "displayName": ["Updated User"],
  "manager": {
    "_id" : "bjensen",
    "displayName" : "Babs Jensen"
  }
 }' \
 https://localhost:8443/api/users/newuser?_prettyPrint=true

{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<timestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "Updated User" ],
  "name" : {
    "givenName" : "Updated",
    "familyName" : "User"
  },
  "manager" : {
    "_id" : "bjensen",
    "_rev" : "<revision>"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 234 567 8910",
    "emailAddress" : "updated.user@example.com"
  }
}
```

```
PS C:\path\to> $Credentials =
[System.Convert]::ToBase64String([System.Text.Encoding]::ASCII.Get
Bytes("admin:password"))
```

```
$Headers = @{
    "Authorization" = "Basic $Credentials"
    "If-Match"      = "*"
}
Invoke-RestMethod `
 -Uri https://localhost:8443/api/users/newuser `
 -Method Put `
 -Headers $Headers `
 -ContentType application/json `
 -Body @"
 {
  "_id": "newuser",
  "_schema":"frapi:opendj:rest2ldap:user:1.0",
  "contactInformation": {
    "telephoneNumber": "+1 234 567 8910",
    "emailAddress": "updated.user@example.com"
  },
  "name": {
    "givenName": "Updated",
    "familyName": "User"
  },
  "displayName": ["Updated User"],
  "manager": {
    "_id" : "bjensen",
    "displayName" : "Babs Jensen"
  }
 }
"@ | ConvertTo-JSON

{
    "_id":  "newuser",
    "_rev":  "<revision>",
    "_schema":  "frapi:opendj:rest2ldap:user:1.0",
    "_meta":  {
                  "created":  "<timestamp>",
                  "lastModified":  "<timestamp>"
              },
    "displayName":  [
                        "Updated User"
                    ],
    "name":  {
                 "givenName":  "Updated",
                 "familyName":  "User"
             },
    "manager":  {
```

```
                    "_id":  "bjensen",
                    "_rev":  "<revision>"
                },
    "contactInformation":  {
                                "telephoneNumber":  "+1 234 567
8910"
                            }
}
```

```
% curl \
 --request PUT \
 --cacert ca-cert.pem \
 --user admin:password \
 --header "Content-Type: application/json" \
 --header "If-Match: *" \
 --data '{
  "_id": "newuser",
  "_schema":"frapi:opendj:rest2ldap:user:1.0",
  "contactInformation": {
    "telephoneNumber": "+1 234 567 8910",
    "emailAddress": "updated.user@example.com"
  },
  "name": {
    "givenName": "Updated",
    "familyName": "User"
  },
  "displayName": ["Updated User"],
  "manager": {
    "_id" : "bjensen",
    "displayName" : "Babs Jensen"
  }
 }' \
 "https://localhost:8443/api/users/newuser?_prettyPrint=true"

{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<timestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "Updated User" ],
  "name" : {
    "givenName" : "Updated",
    "familyName" : "User"
  },
  "manager" : {
    "_id" : "bjensen",
    "_rev" : "<revision>"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 234 567 8910",
    "emailAddress" : "updated.user@example.com"
  }
}
```

Resources are versioned using revision numbers. A revision is specified in the resource's `\_rev` field. The `--header "If-Match: *"` ensures the resource is replaced, regardless of its revision. Alternatively, you can set `--header "If-Match: revision"` to replace only the expected revision of the resource.

For additional details, see DS REST APIs and Update. You can also patch resources instead of replacing them entirely. See Patch.

Delete

Use the REST API to delete the user resource updated in Update:

1. Bash
2. PowerShell
3. Zsh

```
$ curl \
 --request DELETE \
 --cacert ca-cert.pem \
 --user admin:password \
 --header "Content-Type: application/json" \
 https://localhost:8443/api/users/newuser?_prettyPrint=true

{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<timestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "Updated User" ],
  "name" : {
    "givenName" : "Updated",
    "familyName" : "User"
  },
  "manager" : {
    "_id" : "bjensen",
    "_rev" : "<revision>"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 234 567 8910",
    "emailAddress" : "updated.user@example.com"
  }
}
```

```
PS C:\path\to> $Credentials =
[System.Convert]::ToBase64String([System.Text.Encoding]::ASCII.Get
Bytes("admin:password"))
$Headers = @{
    Authorization = "Basic $Credentials"
}
Invoke-RestMethod `
 -Uri https://localhost:8443/api/users/newuser `
 -Method Delete `
 -Headers $Headers `
 -ContentType application/json | ConvertTo-JSON

{
    "_id":  "newuser",
    "_rev":  "<revision>",
    "_schema":  "frapi:opendj:rest2ldap:user:1.0",
    "_meta":  {
                  "created":  "<timestamp>",
                  "lastModified":  "<timestamp>"
              },
    "displayName":  [
                        "Updated User"
                    ],
    "name":  {
                 "givenName":  "Updated",
                 "familyName":  "User"
             },
    "manager":  {
                    "_id":  "bjensen",
                    "_rev":  "<revision>"
                },
    "contactInformation":  {
                              "telephoneNumber":  "+1 234 567
8910"
                           }
}
```

```
% curl \
 --request DELETE \
 --cacert ca-cert.pem \
 --user admin:password \
 --header "Content-Type: application/json" \
 "https://localhost:8443/api/users/newuser?_prettyPrint=true"

{
  "_id" : "newuser",
  "_rev" : "<revision>",
  "_schema" : "frapi:opendj:rest2ldap:user:1.0",
  "_meta" : {
    "created" : "<timestamp>"
  },
  "userName" : "newuser@example.com",
  "displayName" : [ "Updated User" ],
  "name" : {
    "givenName" : "Updated",
    "familyName" : "User"
  },
  "manager" : {
    "_id" : "bjensen",
    "_rev" : "<revision>"
  },
  "contactInformation" : {
    "telephoneNumber" : "+1 234 567 8910",
    "emailAddress" : "updated.user@example.com"
  }
}
```

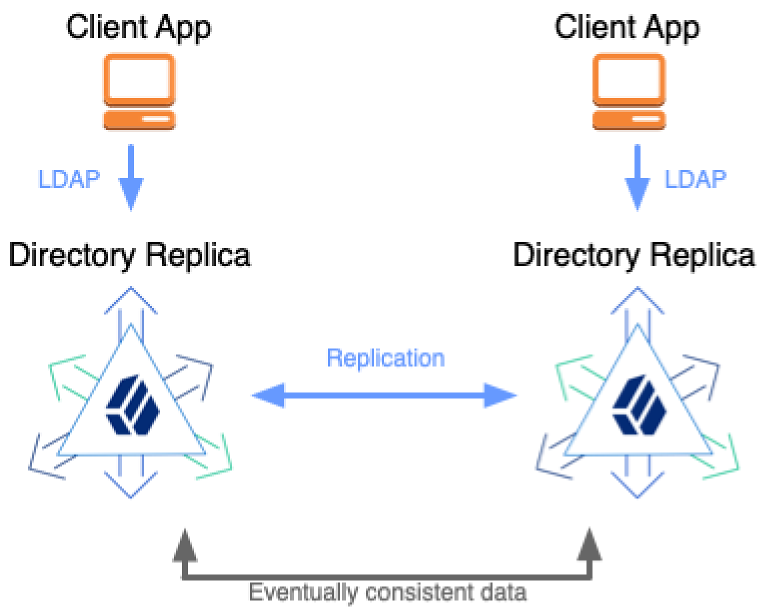
For additional details, see DS REST APIs and Delete.

Learn Replication

Replication provides automatic data synchronization between directory servers. It ensures that all directory servers eventually share a consistent set of directory data.

▼ More about replication

Replication requires two or more directory servers and additional configuration. This page takes you though the setup process quickly, providing commands that you can reuse. It does not explain each command in detail.



For a full discussion of the subject, see Replication.

Add a Replica

High-level steps:

1. Unpack the files for a second directory server in a different folder.
2. Set up the new server as a replica of the first server using the generated `<deployment-key>` from Install DS.

The following example demonstrates the process:

1. Bash
2. PowerShell
3. Zsh

```
# Unpack files for a second, replica server in a different folder:
cd ~/Downloads && unzip ~/Downloads/DS-7.1.8.zip && mv opendj
/path/to/replica

# Set up a second, replica server:
/path/to/replica/setup \
 --serverId second-ds \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDn uid=admin \
 --rootUserPassword password \
 --hostname localhost \
 --ldapPort 11389 \
 --ldapsPort 11636 \
 --adminConnectorPort 14444 \
 --replicationPort 18989 \
 --bootstrapReplicationServer localhost:8989 \
 --profile ds-evaluation \
 --start \
 --acceptLicense
```

```
# Unpack files for a second, replica server in a different folder:
Expand-Archive DS-7.1.8.zip C:\Temp
Rename-Item -Path C:\Temp\opendj -NewName C:\Temp\replica
Move-Item C:\Temp\replica C:\path\to

# Set up a second, replica server:
C:\path\to\replica\setup.bat `
 --serverId second-ds `
 --deploymentKey <deployment-key> `
 --deploymentKeyPassword password `
 --rootUserDn uid=admin `
 --rootUserPassword password `
 --hostname localhost `
 --ldapPort 11389 `
 --ldapsPort 11636 `
 --adminConnectorPort 14444 `
 --replicationPort 18989 \
 --bootstrapReplicationServer locahost:8989 \
 --profile ds-evaluation `
 --start `
 --acceptLicense
```

```
# Unpack files for a second, replica server in a different folder:
cd ~/Downloads && unzip ~/Downloads/DS-7.1.8.zip && mv opendj
/path/to/replica

# Set up a second, replica server:
/path/to/replica/setup \
 --serverId second-ds \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDn uid=admin \
 --rootUserPassword password \
 --hostname localhost \
 --ldapPort 11389 \
 --ldapsPort 11636 \
 --adminConnectorPort 14444 \
 --replicationPort 18989 \
 --bootstrapReplicationServer localhost:8989 \
 --profile ds-evaluation \
 --start \
 --acceptLicense
```

Try Replication

With the new replica set up and started, demonstrate that replication works:

1. Bash
2. PowerShell
3. Zsh

```
# Update a description on the first server:
ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=bjensen,ou=People,dc=example,dc=com \
 --bindPassword hifalutin <<EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Replicate this
EOF

# On the first server, read the description to see the effects of
your change:
ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDn dc=example,dc=com \
 "(cn=Babs Jensen)" \
 description

# On the second server, read the description to see the change has
been replicated:
ldapsearch \
 --hostname localhost \
 --port 11636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDn dc=example,dc=com \
 "(cn=Babs Jensen)" \
 description
```

```
# Update a description on the first server:
New-Item -Path . -Name "mod-desc.ldif" -ItemType "file" -Value @"
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Replicate this
"@

ldapmodify.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn uid=bjensen,ou=People,dc=example,dc=com `
 --bindPassword password `
 mod-desc.ldif

# On the first server, read the description to see the effects of
your change:
ldapsearch.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --baseDn dc=example,dc=com `
 "(cn=Babs Jensen)" `
 description

# On the second server, read the description to see the change has
been replicated:
ldapsearch.bat `
 --hostname localhost `
 --port 11636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --baseDn dc=example,dc=com `
 "(cn=Babs Jensen)" `
 description
```

```
# Update a description on the first server:
ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=bjensen,ou=People,dc=example,dc=com \
 --bindPassword hifalutin <<EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Replicate this
EOF

# On the first server, read the description to see the effects of
your change:
ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDn dc=example,dc=com \
 "(cn=Babs Jensen)" \
 description

# On the second server, read the description to see the change has
been replicated:
```

```
ldapsearch \
 --hostname localhost \
 --port 11636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDn dc=example,dc=com \
 "(cn=Babs Jensen)" \
 description
```

Also demonstrate that replication works despite crashes and network interruptions:

1. Bash
2. PowerShell
3. Zsh

```
# Stop the second server to simulate a network outage or server
crash:
/path/to/replica/bin/stop-ds

# On the first server, update the description again:
ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=bjensen,ou=People,dc=example,dc=com \
 --bindPassword hifalutin <<EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Second server is stopped
EOF

# On the first server, read the description to see the change:
ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDn dc=example,dc=com \
 "(cn=Babs Jensen)" \
 description

# Start the second server again to simulate recovery:
/path/to/replica/bin/start-ds

# On the second server, read the description to check that
replication has resumed:
ldapsearch \
 --hostname localhost \
 --port 11636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDn dc=example,dc=com \
 "(cn=Babs Jensen)" \
 description
```

```
# Stop the second server to simulate a network outage or server
crash:
C:\path\to\replica\bat\stop-ds.bat

# On the first server, update the description again:
New-Item -Path . -Name "mod-desc2.ldif" -ItemType "file" -Value @"
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Second server is stopped
"@

ldapmodify.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn uid=bjensen,ou=People,dc=example,dc=com `
 --bindPassword password `
mod-desc2.ldif

# On the first server, read the description to see the change:
ldapsearch.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --baseDn dc=example,dc=com `
 "(cn=Babs Jensen)" `
 description

# Start the second server again to simulate recovery:
C:\path\to\replica\bat\start-ds.bat

# On the second server, read the description to check that
replication has resumed:
ldapsearch.bat `
 --hostname localhost `
 --port 11636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --baseDn dc=example,dc=com `
 "(cn=Babs Jensen)" `
 description
```

```
# Stop the second server to simulate a network outage or server
crash:
/path/to/replica/bin/stop-ds

# On the first server, update the description again:
ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=bjensen,ou=People,dc=example,dc=com \
 --bindPassword hifalutin <<EOF
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Second server is stopped
EOF

# On the first server, read the description to see the change:
ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDn dc=example,dc=com \
 "(cn=Babs Jensen)" \
 description

# Start the second server again to simulate recovery:
/path/to/replica/bin/start-ds

# On the second server, read the description to check that
replication has resumed:
ldapsearch \
 --hostname localhost \
 --port 11636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDn dc=example,dc=com \
 "(cn=Babs Jensen)" \
 description
```

Unlike some databases, DS replication does not operate in active-passive mode. Instead, you read and write on any running server. Replication replays your changes as soon as possible. Demonstrate this to check your understanding:

1. Stop the first server.

   ▼ *Hint*

   Use the **`stop-ds`** command.

2. Modify an entry on the second server.

   ▼ *Hint*

   For an example, see Modify.

3. Restart the first server.

   ▼ *Hint*

   Use the **`start-ds`** command.

4. Search for the modified entry on the first server to check that replication replays the change.

   ▼ *Hint*

   For an example, see Search.

Notifications

Some applications require notification when directory data updates occur. For example, IDM can sync directory data with another database. Other applications start additional processing when certain updates occur.

Replicated DS directory servers publish an external change log over LDAP. This changelog allows authorized client applications to read changes to directory data:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN cn=changelog \
 --control "ecl:false" \
 "(&)" \
 changes changeLogCookie targetDN
```

```
C:\> ldapsearch.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDN uid=admin `
 --bindPassword password `
 --baseDN cn=changelog `
 --control "ecl:false" `
 "(objectclass=*)" `
 changes changeLogCookie targetDN
```

```
% ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=admin \
 --bindPassword password \
 --baseDN cn=changelog \
 --control "ecl:false" \
 "(&)" \
 changes changeLogCookie targetDN
```

When looking at the output of the command (not shown here), notice that the `changes` values are base64-encoded in LDIF because they include line breaks. You can use the DS **`base64`** command to decode them. For details, see Changelog for Notifications.

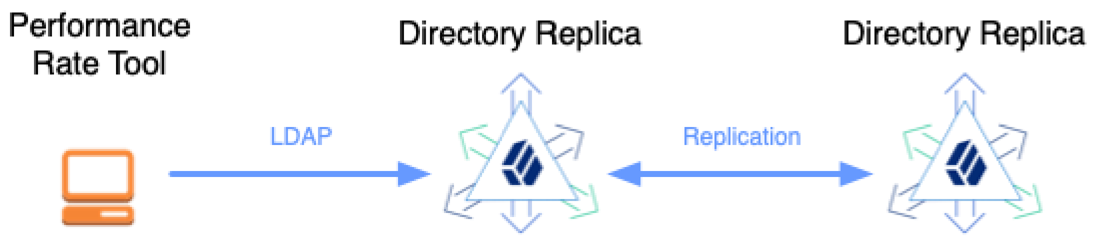
Measure Performance

DS directory servers offer high throughput and low response times for most operations. DS software includes the following command-line tools for measuring performance of common LDAP operations:

- **`addrate`** measures LDAP adds and deletes
- **`authrate`** measures LDAP binds
- **`modrate`** measures LDAP modifications
- **`searchrate`** measures LDAP searches

NOTE

Before trying the examples that follow, work through the previous examples. You should have two directory server replicas running on your local computer, as described in Learn Replication:



Modifications

Measure the LDAP modification rate:

1. Bash
2. PowerShell

3. Zsh

```
# Run modrate for 10 seconds against the first server:
modrate \
 --maxDuration 10 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=bjensen,ou=People,dc=example,dc=com \
 --bindPassword hifalutin \
 --noRebind \
 --numConnections 4 \
 --numConcurrentRequests 4 \
 --targetDn "uid=user.{1},ou=people,dc=example,dc=com" \
 --argument "rand(0,100000)" \
 --argument "randstr(16)" \
 "description:{2}"

# Read number of modify requests on the LDAPS port:
ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=monitor \
 --bindPassword password \
 --baseDN "cn=LDAPS,cn=connection handlers,cn=monitor" \
 "(&)" \
 ds-mon-requests-modify
```

```
# Run modrate for 10 seconds against the first server, and observe
the performance numbers:
modrate.bat `
 --maxDuration 10 `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn uid=bjensen,ou=People,dc=example,dc=com `
 --bindPassword password `
 --noRebind `
 --numConnections 4 `
 --numConcurrentRequests 4 `
 --targetDn "uid=user.{1},ou=people,dc=example,dc=com" `
 --argument "rand(0,100000)" `
 --argument "randstr(16)" `
 "description:{2}"

# Read number of modify requests on the LDAPS port:
ldapsearch.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDN uid=monitor `
 --bindPassword password `
 --baseDN "cn=LDAPS,cn=connection handlers,cn=monitor" `
 "(objectclass=*)" `
 ds-mon-requests-modify
```

```
# Run modrate for 10 seconds against the first server:
modrate \
 --maxDuration 10 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=bjensen,ou=People,dc=example,dc=com \
 --bindPassword hifalutin \
 --noRebind \
 --numConnections 4 \
 --numConcurrentRequests 4 \
 --targetDn "uid=user.{1},ou=people,dc=example,dc=com" \
 --argument "rand(0,100000)" \
 --argument "randstr(16)" \
 "description:{2}"

# Read number of modify requests on the LDAPS port:
ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=monitor \
 --bindPassword password \
 --baseDN "cn=LDAPS,cn=connection handlers,cn=monitor" \
 "(&)" \
 ds-mon-requests-modify
```

When reading the **`modrate`** command output, notice that it shows statistics for throughput (operations/second), response times (milliseconds), and errors/second. If you expect all operations to succeed and yet `err/sec` is not 0.0, the command options are no doubt incorrectly set. For an explanation of the command output, see modrate.

Notice that the monitoring attributes hold similar, alternative statistics.

Searches

Measure the LDAP search rate:

1. Bash
2. PowerShell
3. Zsh

```
# Run searchrate for 10 seconds against the first server:
searchrate \
 --maxDuration 10 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=bjensen,ou=People,dc=example,dc=com \
 --bindPassword hifalutin \
 --noRebind \
 --numConnections 4 \
 --numConcurrentRequests 4 \
 --baseDn "dc=example,dc=com" \
 --argument "rand(0,100000)" \
 "(uid=user.{})"

# Read number of subtree search requests on the LDAPS port:
ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=monitor \
 --bindPassword password \
 --baseDN "cn=LDAPS,cn=connection handlers,cn=monitor" \
 "(&)" \
 ds-mon-requests-search-sub
```

```
# Run searchrate for 10 seconds against the first server:
searchrate.bat `
 --maxDuration 10 `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn uid=bjensen,ou=People,dc=example,dc=com `
 --bindPassword password `
 --noRebind `
 --numConnections 4 `
 --numConcurrentRequests 4 `
 --baseDn "dc=example,dc=com" `
 --argument "rand(0,100000)" `
 "(uid=user.{})"

# Read number of subtree search requests on the LDAPS port:
ldapsearch.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDN uid=monitor `
 --bindPassword password `
 --baseDN "cn=LDAPS,cn=connection handlers,cn=monitor" `
 "(objectclass=*)" `
 ds-mon-requests-search-sub
```

```
# Run searchrate for 10 seconds against the first server:
searchrate \
 --maxDuration 10 \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=bjensen,ou=People,dc=example,dc=com \
 --bindPassword hifalutin \
 --noRebind \
 --numConnections 4 \
 --numConcurrentRequests 4 \
 --baseDn "dc=example,dc=com" \
 --argument "rand(0,100000)" \
 "(uid=user.{})"

# Read number of subtree search requests on the LDAPS port:
ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=monitor \
 --bindPassword password \
 --baseDN "cn=LDAPS,cn=connection handlers,cn=monitor" \
 "(&)" \
 ds-mon-requests-search-sub
```

Notice that **`searchrate`** command output resembles that of the **`modrate`** command. The **`searchrate`** output also indicates how many entries each search returned. For an explanation of the command output, see searchrate.

Check Replication

After running the performance tools, check that both replicas are up to date. The following example uses monitoring metrics to check that replication delay is zero on each replica:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=monitor \
 --bindPassword password \
 --baseDN cn=monitor \
 "(ds-mon-current-delay=*)" \
 ds-mon-current-delay

dn: ds-mon-domain-
name=dc=example\,dc=com,cn=replicas,cn=replication,cn=monitor
ds-mon-current-delay: 0

dn: ds-mon-server-id=second-ds,cn=remote replicas,ds-mon-domain-
name=dc=example\,dc=com,cn=replicas,cn=replication,cn=monitor
ds-mon-current-delay: 0
```

```
PS C:\path\to> ldapsearch.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDN uid=monitor `
 --bindPassword password `
 --baseDN cn=monitor `
 "(ds-mon-current-delay=*)" `
 ds-mon-current-delay

dn: ds-mon-domain-
name=dc=example\,dc=com,cn=replicas,cn=replication,cn=monitor
ds-mon-current-delay: 0

dn: ds-mon-server-id=second-ds,cn=remote replicas,ds-mon-domain-
name=dc=example\,dc=com,cn=replicas,cn=replication,cn=monitor
ds-mon-current-delay: 0
```

```
% ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDN uid=monitor \
 --bindPassword password \
 --baseDN cn=monitor \
 "(ds-mon-current-delay=*)" \
 ds-mon-current-delay

dn: ds-mon-domain-
name=dc=example\,dc=com,cn=replicas,cn=replication,cn=monitor
ds-mon-current-delay: 0

dn: ds-mon-server-id=second-ds,cn=remote replicas,ds-mon-domain-
name=dc=example\,dc=com,cn=replicas,cn=replication,cn=monitor
ds-mon-current-delay: 0
```

Learn Access Control

Until now, you have used the evaluation setup profile. The evaluation profile makes it easy to access Example.com data. It helps you learn and demonstrate directory services without explicitly granting access after server setup.

In a production directory service where security is important, access is under tighter control. In most cases, access is denied by default to prevent accidental information leaks. You must explicitly grant access where required. To grant access, use access control instructions (ACIs).

NOTE

The sample ACIs described here demonstrate some but not all ACI features.

For details, see Access Control.

About ACIs

ACIs are implemented as *operational LDAP attributes*. An operational attribute is not meant to store application data, but to influence server behavior. Operational attributes are often left hidden from normal users. A server does not return operational attributes on an entry unless explicitly requested.

Each ACI influences server behavior by indicating:

- Which directory data it targets
- Which permissions it allows or denies
- Which users or groups it applies to
- Under which conditions (time, network origin, connection security, user properties) it applies

▼ Example ACI with explanation

The following example ACI gives users access to change their own passwords:

```
aci: (targetattr = "authPassword || userPassword")
 (version 3.0;acl "Allow users to change their own passwords";
 allow (write)(userdn = "ldap:///self");)
```

Consider the characteristics of this ACI attribute:

Target Entries and Scope

The target entries and scope for this ACI are implicit.

The default target is the entry with this `aci` attribute.

The default scope includes the target entry and all its subordinates.

In other words, if you set this ACI on `ou=People,dc=example,dc=com`, it affects all users under that base entry. For example, Babs Jensen, `uid=bjensen,ou=People,dc=example,dc=com`, can set her own password.

Target Attributes

This ACI affects operations on either of the standard password attributes: `(targetattr = "authPassword || userPassword")`.

The ACI only has an effect when an operation targets either `authPassword` or `userPassword`, and any subtypes of those attribute types.

Permissions

This ACI affects only operations that change affected attributes: `allow (write)`.

If this is the only ACI that targets password attributes, users have access to change their own passwords, but they do not have access to *read* passwords.

Subjects

This ACI has an effect when the target entry is the same as the bind DN: `(userdn = "ldap:///self")`.

This means that the user must have authenticated to change their password.

Documentation

The wrapper around the permissions and subjects contains human-readable documentation about the ACI: `(version 3.0;acl "Allow users to change their own passwords"; … ;)`.

Version 3.0 is the only supported ACI version.

Conditions

This ACI does not define any conditions. It applies all the time, for connections from all networks, and so forth.

Server configuration settings can further constrain how clients connect. Such constraints are not specified by this ACI, however.

Use ACIs

To write ACI attributes:

- A user must have the `modify-acl` administrative privilege.

  Privileges are server configuration settings that control access to administrative operations.
- An ACI must give the user permission to change `aci` attributes.

IMPORTANT

By default, only the directory superuser has the right to add, delete, or modify ACI attributes. In fact, the directory superuser has a privilege, `bypass-acl`, that allows the account to perform operations without regard to ACIs.

Any account with permissions to change ACIs is dangerous, because the power can be misused. The user with permissions to change ACIs can give themselves full access to all directory data in their scope.

Prepare to use the examples:

▼ Stop running servers

Use each server's **`stop-ds`** command to stop any DS servers running on your computer.

This lets the new server use ports that might already be in use by another server.

▼ Get sample data

1. Download the Example.ldif file, shown in the following listing:

   ▼ *Show listing*

```
#
# Copyright 2020-2021 ForgeRock AS. All Rights Reserved
#
# Use of this code requires a commercial software license
with ForgeRock AS.
# or with one of its affiliates. All use shall be
exclusively subject
# to such license between the licensee and ForgeRock AS.
#
dn: dc=example,dc=com
objectClass: domain
```

```
objectClass: top
dc: example

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

dn: ou=Self Service,ou=Groups,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
description: Groups that authenticated users can manage on
their own
ou: Self Service

dn: ou=People,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
description: Description on ou=People
ou: People

dn: uid=ACI Admin,ou=People,dc=example,dc=com
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: top
cn: ACI Admin
givenName: ACI
mail: aci-admin@example.com
ou: People
sn: Admin
uid: ACI Admin
userPassword: 5up35tr0ng

dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: top
cn: Babs Jensen
givenName: Barbara
mail: bjensen@example.com
ou: People
sn: Jensen
```

```
uid: bjensen
userPassword: 5up35tr0ng
```

2. Save the file to your computer’s temporary directory, such as `/tmp` or `C:\Temp` .

▼ Install server with secure settings

1. Unzip the DS server `.zip` file into the folder where you want to install the server.
2. Set up the directory server using the LDIF you downloaded.

   Set up the server without the evaluation setup profile, *so the access control settings are secure by default.* The default password policies require stronger passwords. The configuration grants very little access to regular users. Only `uid=admin` has access to the data:

   1. Bash
   2. PowerShell
   3. Zsh

```
$ /path/to/opendj/setup \
 --serverId learn-acis \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDn uid=admin \
 --rootUserPassword str0ngAdm1nPa55word \
 --hostname localhost \
 --ldapPort 1389 \
 --ldapsPort 1636 \
 --httpsPort 8443 \
 --adminConnectorPort 4444 \
 --acceptLicense
$ dsconfig \
 create-backend \
 --backend-name exampleData \
 --type je \
 --set enabled:true \
 --set base-dn:dc=example,dc=com \
 --offline \
 --no-prompt
$ import-ldif \
 --backendId exampleData \
 --ldifFile /tmp/Example.ldif \
 --offline
$ start-ds --quiet
```

```
PS C:\path\to> C:\path\to\opendj\setup.bat `
 --serverId learn-acis `
 --deploymentKey $DEPLOYMENT_KEY `
 --deploymentKeyPassword password `
 --rootUserDn uid=admin `
 --rootUserPassword str0ngAdm1nPa55word `
 --hostname localhost `
 --ldapPort 1389 `
 --ldapsPort 1636 `
 --httpsPort 8443 `
 --adminConnectorPort 4444 `
 --acceptLicense
PS C:\path\to> C:\path\to\opendj\bat\dsconfig.bat `
 create-backend `
 --backend-name exampleData `
 --type je `
 --set enabled:true `
 --set base-dn:dc=example,dc=com `
 --offline `
 --no-prompt
PS C:\path\to> C:\path\to\opendj\bat\import-ldif.bat `
 --backendId exampleData `
 --ldifFile C:\Temp\Example.ldif `
 --offline
PS C:\path\to> C:\path\to\opendj\bat\start-ds.bat --quiet
```

```
% /path/to/opendj/setup \
 --serverId learn-acis \
 --deploymentKey $DEPLOYMENT_KEY \
 --deploymentKeyPassword password \
 --rootUserDn uid=admin \
 --rootUserPassword str0ngAdm1nPa55word \
 --hostname localhost \
 --ldapPort 1389 \
 --ldapsPort 1636 \
 --httpsPort 8443 \
 --adminConnectorPort 4444 \
 --acceptLicense
% dsconfig \
 create-backend \
 --backend-name exampleData \
 --type je \
 --set enabled:true \
 --set base-dn:dc=example,dc=com \
 --offline \
 --no-prompt
% import-ldif \
 --backendId exampleData \
 --ldifFile /tmp/Example.ldif \
 --offline
% start-ds --quiet
```

▼ Grant ACI admin access

Grant the `ACI Admin` user access to modify ACIs:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=admin \
 --bindPassword str0ngAdm1nPa55word << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "aci") (version 3.0;acl "ACI Admin can manage
ACI attributes";
 allow (write) userdn = "ldap:///uid=ACI
Admin,ou=People,dc=example,dc=com";)

dn: uid=ACI Admin,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: modify-acl
EOF
```

```
PS C:\path\to> New-Item -Path . -Name "aci-admin.ldif" -ItemType
"file" -Value @"
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "aci") (version 3.0;acl "ACI Admin can manage
ACI attributes";
 allow (write) userdn = "ldap:///uid=ACI
Admin,ou=People,dc=example,dc=com";)

dn: uid=ACI Admin,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: modify-acl
"@
PS C:\path\to> ldapmodify.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin
`
 --bindDn uid=admin `
 --bindPassword str0ngAdm1nPa55word `
 aci-admin.ldif
```

```
% ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn uid=admin \
 --bindPassword str0ngAdm1nPa55word << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "aci") (version 3.0;acl "ACI Admin can manage
ACI attributes";
 allow (write) userdn = "ldap:///uid=ACI
Admin,ou=People,dc=example,dc=com";)

dn: uid=ACI Admin,ou=People,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: modify-acl
EOF
```

▼ (Optional) Try LDAP examples

Try examples from Learn LDAP.

You find that Babs Jensen does not have the access that she had with the evaluation setup profile. For production servers, the best practice is to grant access only when required.

Examples

Prepare to use the examples before trying them. The `ACI Admin` account must have access to manage ACIs. After you add an example ACI, test users' access. For inspiration, see the examples in Learn LDAP.

ACI syntax is powerful, and sometimes difficult to get right. For details, see Access Control.

ACI: Access Own Entry

The following example grants authenticated users access to read their own entry, and modify some attributes:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "*") (version 3.0;acl "Users can read their
entries";
 allow (read, search, compare) (userdn = "ldap:///self");)
-
add: aci
aci: (targetattr = "authPassword || description || displayName ||
homePhone ||
 jpegPhoto || preferredLanguage || userPassword")
 (version 3.0;acl "Self-service modifications for basic
attributes";
```

```
 allow (write) (userdn = "ldap:///self");)
EOF
```

```
PS C:\path\to> New-Item -Path . -Name "self-access.ldif" -ItemType
"file" -Value @"
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "*") (version 3.0;acl "Users can read their
entries";
 allow (read, search, compare) (userdn = "ldap:///self");)
-
add: aci
aci: (targetattr = "authPassword || description || displayName ||
homePhone ||
 jpegPhoto || preferredLanguage || userPassword")
 (version 3.0;acl "Self-service modifications for basic
attributes";
 allow (write) (userdn = "ldap:///self");)
"@
PS C:\path\to> ldapmodify.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" `
 --bindPassword 5up35tr0ng `
 self-access.ldif
```

```
% ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "*") (version 3.0;acl "Users can read their
entries";
 allow (read, search, compare) (userdn = "ldap:///self");)
```

```
-
add: aci
aci: (targetattr = "authPassword || description || displayName ||
homePhone ||
 jpegPhoto || preferredLanguage || userPassword")
 (version 3.0;acl "Self-service modifications for basic
attributes";
 allow (write) (userdn = "ldap:///self");)
EOF
```

In this example, the list of attributes that users can read includes all user attributes. The list that users can modify is limited. Other attributes might be governed by other applications. For example, a user's manager might only be changed through an HR system. Perhaps the IT department is responsible for all changes to email addresses.

ACI: Access SubSchemaSubEntry Attribute

The `subSchemaSubEntry` attribute indicates the entry holding the LDAP schema definitions that apply to the current entry. Many applications retrieve this attribute, and the associated schema, to properly display or validate attribute values.

The following example demonstrates how to grant access to read this attribute on directory entries:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "subSchemaSubEntry")
 (version 3.0;acl "Authenticated users can read
subSchemaSubEntry";
 allow (read, search, compare) (userdn = "ldap:///all");)
EOF
```

```
PS C:\path\to> New-Item -Path . -Name "subSchemaSubentry-
access.ldif" -ItemType "file" -Value @"
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "subSchemaSubEntry")
 (version 3.0;acl "Authenticated users can read
subSchemaSubEntry";
 allow (read, search, compare) (userdn = "ldap:///all");)
"@
PS C:\path\to> ldapmodify.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" `
 --bindPassword 5up35tr0ng `
 subSchemaSubentry-access.ldif
```

```
% ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "subSchemaSubEntry")
 (version 3.0;acl "Authenticated users can read
subSchemaSubEntry";
 allow (read, search, compare) (userdn = "ldap:///all");)
EOF
```

ACI: Manage Group Membership

For some static groups, you might choose to let users manage their own memberships. The following example lets members of self-service groups manage their own membership:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: ou=Self Service,ou=Groups,dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "member") (version 3.0;acl "Self registration";
 allow (selfwrite) (userdn =
"ldap:///uid=*,ou=People,dc=example,dc=com");)
EOF
```

```
PS C:\path\to> New-Item -Path . -Name "self-service-groups.ldif" -
ItemType "file" -Value @"
dn: ou=Self Service,ou=Groups,dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "member") (version 3.0;acl "Self registration";
 allow (selfwrite) (userdn =
"ldap:///uid=*,ou=People,dc=example,dc=com");)
"@
PS C:\path\to> ldapmodify.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" `
 --bindPassword 5up35tr0ng `
 self-service-groups.ldif
```

```
% ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: ou=Self Service,ou=Groups,dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "member") (version 3.0;acl "Self registration";
 allow (selfwrite) (userdn =
"ldap:///uid=*,ou=People,dc=example,dc=com");)
EOF
```

The `selfwrite` permission is for adding or deleting one's own DN from a group.

ACI: Manage Self-Service Groups

This example lets users create and delete self-managed groups:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: ou=Self Service,ou=Groups,dc=example,dc=com
changetype: modify
add: aci
aci: (targattrfilters="add=objectClass:
(objectClass=groupOfNames)")
 (version 3.0; acl "Users can create self-service groups";
 allow (add) (userdn =
"ldap:///uid=*,ou=People,dc=example,dc=com");)
-
add: aci
aci: (version 3.0; acl "Owner can delete self-service groups";
 allow (delete) (userattr = "owner#USERDN");)
EOF
```

```
PS C:\path\to> New-Item -Path . -Name "self-managed-groups.ldif" -
ItemType "file" -Value @"
dn: ou=Self Service,ou=Groups,dc=example,dc=com
changetype: modify
add: aci
aci: (targattrfilters="add=objectClass:
(objectClass=groupOfNames)")
 (version 3.0; acl "Users can create self-service groups";
 allow (add) (userdn =
"ldap:///uid=*,ou=People,dc=example,dc=com");)
-
add: aci
aci: (version 3.0; acl "Owner can delete self-service groups";
 allow (delete) (userattr = "owner#USERDN");)
"@
PS C:\path\to> ldapmodify.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" `
 --bindPassword 5up35tr0ng `
 self-managed-groups.ldif
```

```
% ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: ou=Self Service,ou=Groups,dc=example,dc=com
changetype: modify
add: aci
aci: (targattrfilters="add=objectClass:
(objectClass=groupOfNames)")
 (version 3.0; acl "Users can create self-service groups";
 allow (add) (userdn =
"ldap:///uid=*,ou=People,dc=example,dc=com");)
-
add: aci
aci: (version 3.0; acl "Owner can delete self-service groups";
 allow (delete) (userattr = "owner#USERDN");)
EOF
```

ACI: Full Access

The following ACI grants Babs Jensen permission to perform all LDAP operations, allowing her full administrator access to the directory data under `dc=example,dc=com`. Babs can read and write directory data, rename and move entries, and use proxied authorization. Some operations also require administrative privileges not shown in this example:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "* || +") (version 3.0;acl "Babs has full
access";
 allow (all, export, import, proxy) (userdn =
"ldap:///uid=bjensen,ou=People,dc=example,dc=com");)
EOF
```

```
PS C:\path\to> New-Item -Path . -Name "full-access.ldif" -ItemType
"file" -Value @"
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "* || +") (version 3.0;acl "Babs has full
access";
 allow (all, export, import, proxy) (userdn =
"ldap:///uid=bjensen,ou=People,dc=example,dc=com");)
"@
PS C:\path\to> ldapmodify.bat `
```

```
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" `
 --bindPassword 5up35tr0ng `
 full-access.ldif
```

```
% ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "* || +") (version 3.0;acl "Babs has full
access";
 allow (all, export, import, proxy) (userdn =
"ldap:///uid=bjensen,ou=People,dc=example,dc=com");)
EOF
```

`(targetattr = "* || +")` permits access to all user attributes and all operational attributes. `allow (all, import, export, proxy)` permits all user operations, modify DN operations, and proxied authorization. Notice that `all` does not allow modify DN and proxied authorization.

ACI: Anonymous Reads and Searches

In LDAP, an anonymous user is one who does not provide bind credentials. By default, most setup profiles only allow anonymous access to read information about the server's capabilities, or before using the StartTLS operation to get a secure connection before providing credentials.

Unless you set up the server with the evaluation profile, anonymous users cannot read application data by default. You can grant them access, however. First, change the global configuration to allow unauthenticated requests. Second, add an ACI to grant access to the entries.

The following command changes the global configuration property, `unauthenticated-requests-policy`, to allow unauthenticated requests:

1. Bash
2. PowerShell
3. Zsh

```
$ dsconfig \
 set-global-configuration-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword str0ngAdm1nPa55word \
 --set unauthenticated-requests-policy:allow \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

```
PS C:\path\to> dsconfig.bat `
 set-global-configuration-prop `
 --hostname localhost `
 --port 4444 `
 --bindDN uid=admin `
 --bindPassword str0ngAdm1nPa55word `
 --set unauthenticated-requests-policy:allow `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --no-prompt
```

```
% dsconfig \
 set-global-configuration-prop \
 --hostname localhost \
 --port 4444 \
 --bindDN uid=admin \
 --bindPassword str0ngAdm1nPa55word \
 --set unauthenticated-requests-policy:allow \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --no-prompt
```

This ACI makes all user attributes in `dc=example,dc=com` data (except passwords) world-readable:

1. Bash
2. PowerShell

3. Zsh

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr != "authPassword || userPassword") (version
3.0;acl "Anonymous read-search access";
 allow (read, search, compare) (userdn = "ldap:///anyone");)
EOF
```

```
PS C:\path\to> New-Item -Path . -Name "anon-access.ldif" -ItemType
"file" -Value @"
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr != "authPassword || userPassword") (version
3.0;acl "Anonymous read-search access";
 allow (read, search, compare) (userdn = "ldap:///anyone");)
"@
PS C:\path\to> ldapmodify.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" `
 --bindPassword 5up35tr0ng `
 anon-access.ldif
```

```
% ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr != "authPassword || userPassword") (version
3.0;acl "Anonymous read-search access";
 allow (read, search, compare) (userdn = "ldap:///anyone");)
EOF
```

Notice that `ldap:///anyone` designates anonymous users and authenticated users. Do not confuse that with `ldap:///all`, which designates authenticated users only.

ACI: Permit Insecure Access Over Loopback Only

This ACI uses IP address and Security Strength Factor subjects to prevent insecure remote access to `dc=example,dc=com` data. In most cases, you explicitly grant permission with `allow`, making it easier to understand and to explain why the server permits a given operation. This demonstrates one use case where it makes sense to *deny* permission:

1. Bash
2. PowerShell
3. Zsh

```
$ ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "* || +") (version 3.0;acl "Restrict insecure
LDAP to the loopback address";
 deny (all) (ip != "127.0.0.1" and ssf <= "1");)
EOF
```

```
PS C:\path\to> New-Item -Path . -Name "deny-cleartext.ldif" -
ItemType "file" -Value @"
```

```
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "* || +") (version 3.0;acl "Restrict cleartext
LDAP to the loopback address";
 deny (all) (ip != "127.0.0.1" and ssf <= "1");)
"@
PS C:\path\to> ldapmodify.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore C:\path\to\opendj\config\keystore `
 --trustStorePassword:file C:\path\to\opendj\config\keystore.pin `
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" `
 --bindPassword 5up35tr0ng `
 deny-cleartext.ldif
```

```
% ldapmodify \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --bindDn "uid=ACI Admin,ou=People,dc=example,dc=com" \
 --bindPassword 5up35tr0ng << EOF
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "* || +") (version 3.0;acl "Restrict insecure
LDAP to the loopback address";
 deny (all) (ip != "127.0.0.1" and ssf <= "1");)
EOF
```

- `ssf = 1` means that TLS is configured without a cipher. The server verifies integrity using packet checksums, but all content is sent in plain text.
- `ssf = 0` means that the content is sent plain text with no connection security.

About Directories

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

Where a directory differs from a paper dictionary or phone book is in how entries are indexed. Dictionaries typically have one index, which is words in alphabetical order. Phone books, have one index as well, which is names in alphabetical order. Directories' entries, however, are often indexed for multiple attributes, including names, user identifiers, email addresses, and telephone numbers. This means you can look up a directory account by the user’s name, their user identifier, their email address, or their telephone number, for example.

ForgeRock Directory Services are based on the Lightweight Directory Access Protocol (LDAP). Nearly all of what follows is an introduction to LDAP.

ForgeRock Directory Services also provide RESTful HTTP access to directory data. As a directory user, you will find it useful to understand the underlying LDAP model even if most users are accessing the directory over HTTP rather than LDAP.

History

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

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

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

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

As organizations rolled out larger and larger directories serving more and more applications, they discovered the need for high availability and fast updates. Around the year 2000, directories began to support multi-master replication; that is, replication with multiple read-write servers. The organizations with the very largest directories became concerned about replicating so many changes.

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

LDAP Data

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

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

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

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

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

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

1. Bash
2. PowerShell

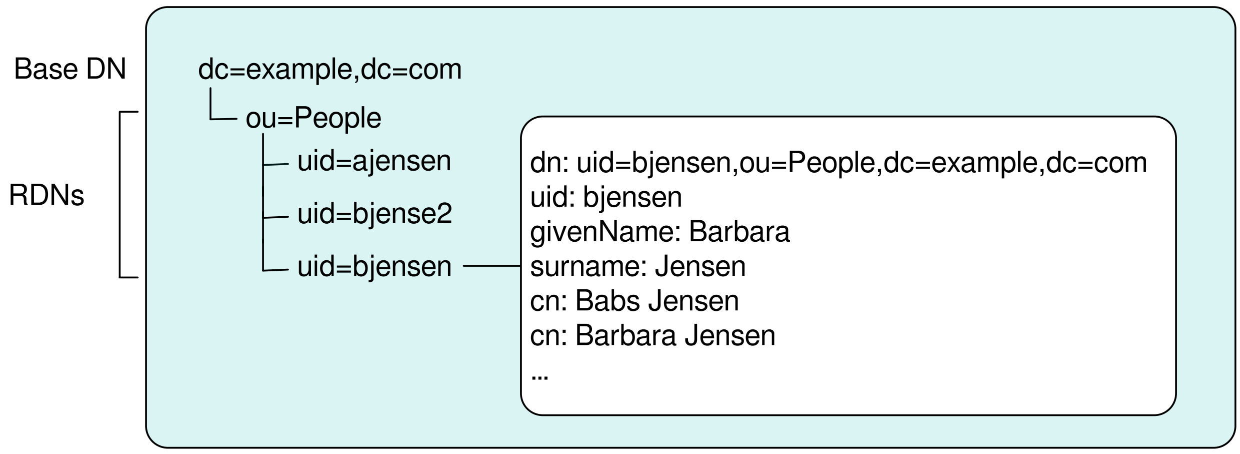
```
$ ldapsearch \
 --hostname localhost \
 --port 1636 \
 --useSsl \
 --usePkcs12TrustStore /path/to/opendj/config/keystore \
 --trustStorePassword:file /path/to/opendj/config/keystore.pin \
 --baseDN dc=example,dc=com \
 "(uid=escape)"

dn: cn=DN Escape Characters \" # \+ \, \; \< = \>
\\,dc=example,dc=com
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: top
givenName: DN Escape Characters
uid: escape
cn: DN Escape Characters " # + , ; < = > \
sn: " # + , ; < = > \
mail: escape@example.com
```

```
PS C:\path\to> ldapsearch.bat `
 --hostname localhost `
 --port 1636 `
 --useSsl `
 --usePkcs12TrustStore /path/to/opendj/config/keystore `
 --trustStorePassword:file /path/to/opendj/config/keystore.pin `
 --baseDN dc=example,dc=com `
 "(uid=escape)"
```

```
dn: cn=DN Escape Characters \" # \+ \, \; \< = \>
\\,dc=example,dc=com
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: top
givenName: DN Escape Characters
uid: escape
cn: DN Escape Characters " # + , ; < = > \
sn: " # + , ; < = > \
mail: escape@example.com
```

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



Barbara Jensen’s entry is located under an entry with DN `ou=People,dc=example,dc=com`, an organizational unit and parent entry for the people at Example.com. The `ou=People` entry is located under the entry with DN `dc=example,dc=com`, the base entry for Example.com. DC is an acronym for *Domain Component*. The directory has other base entries, such as `cn=config`, under which the configuration is accessible through LDAP.

A directory can serve multiple organizations, too. You might find `dc=example,dc=com`, `dc=mycompany,dc=com`, and `o=myOrganization` in the same LDAP directory. Therefore, when you look up entries, you specify the base DN to look under in the same way you need to know whether to look in the New York, Paris, or Tokyo phone book to find a telephone number.

The root entry for the directory, technically the entry with DN `""` (the empty string), is called the *root DSE*. It contains information about what the server supports, including the other base DNs it serves.

A directory server stores two kinds of attributes in a directory entry: *user attributes* and *operational attributes*. User attributes hold the information for users of the directory. All attributes shown in the entry above are user attributes. Operational attributes hold information used by the directory itself. Examples of operational attributes include `entryUUID`, `modifyTimestamp`, and `subschemaSubentry`.

When an LDAP search operation finds an entry in the directory, the directory server returns all the visible user attributes unless the search request restricts the list of attributes by specifying those attributes explicitly. The directory server does not, however, return any operational attributes unless the search request specifically asks for them.

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

Communication

In some client/server applications, like web browsing, a connection is set up and torn down for each client request.

LDAP has a different model. In LDAP, the client application connects to the server and authenticates. The client then requests any number of operations, perhaps processing results in between requests. The client finally disconnects when done, potentially days later.

The standard operations are as follows:

Bind (authenticate)

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

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

Search (lookup)

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

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

Modify
After binding, the client can request that the server change one or more attribute values on an entry. Often administrators do not allow clients to change directory data, so allow appropriate access for client application if they have the right to update data.

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

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

Modify DN
After binding, the client can request that the server change the distinguished name of the entry. In other words, this renames the entry or moves it to another location. For example, if Barbara changes her unique identifier from `bjensen` to something else, her DN would have to change. For another example, if you decide to consolidate `ou=Customers` and `ou=Employees` under `ou=People` instead, all the entries underneath must change distinguished names.

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

Unbind
When done making requests, the client can request an unbind operation to end the LDAP session.

Abandon
When a request seems to be taking too long to complete, or when a search request returns many more matches than desired, the client can send an abandon request to the server to drop the operation in progress.

Controls and Extensions

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

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

One special search operation that DS servers support is Persistent Search. The client application sets up a Persistent Search to continue receiving new results whenever changes are made to data that is in the scope of the search, using the search as a form of change notification. Persistent Searches are intended to remain connected permanently, though they can be idle for long periods of time.

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

For the list of supported LDAP controls, see Supported LDAP Controls.

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

For the list of supported LDAP extended operations, see Supported LDAP Extended Operations.

Indexes

Directories have indexes for multiple attributes. By default, DS does not let normal users perform searches that are not indexed, because such searches mean DS servers have to scan an entire directory database when looking for matches.

As directory administrator, part of your responsibility is making sure directory data is properly indexed. DS software provides tools for building and rebuilding indexes, for verifying indexes, and for evaluating how well indexes are working.

For help better understanding and managing indexes, read Indexes.

Schema

Some databases are designed to hold huge amounts of data for a particular application. Although such databases can support multiple applications, data organization depends on the applications served.

In contrast, directories are designed for shared, centralized services. Although the first guides to deploying directory services suggested taking inventory of all the applications that would access the directory, today many directory administrators do not even know how many applications use their services. The shared, centralized nature of directory services fosters interoperability in practice. It has helped directory services be successful in the long term.

Part of what makes this possible is the shared model of directory user information, in particular the LDAP schema. LDAP schema defines what the directory can contain. This means that directory entries are not arbitrary data, but tightly codified objects whose attributes are completely predictable from publicly readable definitions. Many schema definitions are in fact standard. They are the same not just across a directory service but across different directory services.

At the same time, unlike some databases, LDAP schema and the data it defines can be extended on the fly while the service is running. LDAP schema is accessible over LDAP. One attribute of every entry is its set of `objectClass` values. This gives you as administrator great flexibility in adapting your directory service to store new data without losing or changing the structure of existing data, and without ever stopping your directory service.

For a closer look, see LDAP Schema.

Access Control

Directory services support fine-grained access control.

As directory administrator, you can control who has access to what data when, how, where and under what conditions by using access control instructions (ACI). You can allow some directory operations and not others. You can scope access control from the whole directory service down to individual attributes on directory entries. You can specify when, from what host or IP address, and the encryption strength required for an operation.

As ACIs are stored on entries in the directory, you can update access controls while the service is running, and even delegate that control to client applications. DS software combines the strengths of ACIs with separate administrative privileges to help you secure access to directory data.

For more information, read Access Control.

Replication

DS replication consists of copying each update to the directory service to multiple directory servers. This brings both redundancy, in the case of network partitions or of crashes, and scalability for read operations. Most directory deployments involve multiple servers replicating together.

When you have replicated servers, all of which are writable, you can have replication conflicts. What if, for example, there is a network outage between two replicas, and meanwhile two different values are written to the same attribute on the same entry on the two replicas?

In nearly all cases, DS replication can resolve these situations automatically without involving you, the directory administrator. This makes your directory service resilient and safe even in the unpredictable real world.

One counterintuitive aspect of replication is that although you add directory *read* capacity by adding replicas to your deployment, you do not add directory *write* capacity by adding replicas. Each write operation must be replayed everywhere. As a result, if you have N servers, you have N write operations to replay.

Replication is also *loosely consistent*. Loosely consistent means that directory data will eventually converge to be the same everywhere, but it will not necessarily be the same everywhere at all times, or even at any time. Client applications sometimes get this wrong when they write to a pool of load balanced directory servers, immediately read back what they wrote, and are surprised that it is not the same. If your users are complaining about this, consider using a directory proxy server to mitigate their poor practices.

DSMLv2

Directory Services Markup Language (DSMLv2) v2.0 became a standard in 2001. DSMLv2 describes directory data and basic directory operations in XML format, so they can be carried in Simple Object Access Protocol (SOAP) messages. DSMLv2 further allows clients to batch multiple operations together in a single request, to be processed either in sequential order or in parallel.

DS software provides support for DSMLv2 as a DSML gateway, which is a servlet that connects to any standard LDAPv3 directory. DSMLv2 opens basic directory services to SOAP-based web services and service oriented architectures.

To set up DSMLv2 access, see Install a DSML Gateway.

HTTP Access

DS software can expose directory data as JSON resources over HTTP to REST clients, providing easy access to directory data for developers who are not familiar with LDAP. RESTful access depends on a configuration that describes how the JSON representation maps to LDAP entries.

Although client applications have no need to understand LDAP, the underlying implementation still uses the LDAP model for its operations. The mapping adds some overhead.

Furthermore, depending on the configuration, individual JSON resources can require multiple LDAP operations. For example, an LDAP user entry represents `manager` as a

DN (of the manager's entry). The same manager might be represented in JSON as an object holding the manager's user ID and full name, in which case the software must look up the manager's entry to resolve the mapping for the manager portion of the JSON resource, in addition to looking up the user's entry. As another example, suppose a large group is represented in LDAP as a set of 100,000 DNs. If the JSON resource is configured so that a member is represented by its name, then listing that resource would involve 100,000 LDAP searches to translate DNs to names.

A primary distinction between LDAP entries and JSON resources is that LDAP entries hold sets of attributes values, whereas JSON resources are documents containing arbitrarily nested objects. As LDAP data is governed by schema, almost no LDAP objects are arbitrary collections of data. (LDAP has the object class `extensibleObject`, but its should be used sparingly.) JSON resources can hold arrays, ordered collections that can contain duplicates. LDAP attributes are sets, unordered collections without duplicates. For most directory and identity data, these distinctions do not matter. You are likely to run into them, however, if you try to turn your directory into a document store for arbitrary JSON resources.

Despite some extra cost in terms of system resources, exposing directory data over HTTP can unlock directory services for a new generation of applications. The configuration provides flexible mapping, so that you can configure views that correspond to how client applications need to see directory data.

DS software also give you a deployment choice for HTTP access. You can deploy the REST to LDAP gateway, which is a servlet that connects to any standard LDAPv3 directory, or you can activate the HTTP connection handler on a server to allow direct and more efficient HTTP and HTTPS access.

Deployment

This page serves as an introduction. When you have understood enough of the concepts to build the directory services that you want to deploy, you must still build a prototype and test it before you roll out shared, centralized services for your organization.

Start with Deployment when beginning your project.

Best Practices

Follow these best practices for writing effective, maintainable, high-performance directory client applications.

Authenticate Correctly

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

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

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

Applications can use simple, password-based authentication. When using password-based authentication, use secure connections to protect credentials over the network. For applications, prefer certificate-based authentication if possible.

Reuse Connections

LDAP is a stateful protocol. You authenticate (bind), you perform operations, you unbind. The server maintains a context that lets it make authorization decisions concerning your requests. Therefore, reuse connections whenever possible.

Because LDAP supports asynchronous requests, it is normal and expected to make multiple requests over the same connection. Your application can share a pool of connections to avoid the overhead of setting them up and tearing them down.

Check Connection Health

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

When you maintain a particularly long-lived connection, such as a connection for a persistent search, periodically perform a health check to maintain the connection operational.

A health check involves reading or writing an attribute on a well-known entry in your data. It can serve the purposes of maintaining the connection operational, and of verifying access to your data. A success result for a read indicates that the data is available, and the application can read it. A success result for a write indicates that the data is available, and the application can write to it. The exact check to perform depends on how your application uses the directory. Under some circumstances, your data might be temporarily read-only, for example.

When using a connection timeout, take care not to set the timeout so low that long operations, such as unindexed searches, fail to complete before the timeout.

Request Exactly What You Need All At Once

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

For example, if you require `mail` and `cn`, then specify both attributes in your search request.

Use Specific LDAP Filters

The difference in results between a general filter `(mail=*@example.com)`, and a good, specific filter like `(mail=user@example.com)` can be huge numbers of entries and enormous amounts of processing time, both for the directory server that has to return search results, and for your application that has to sort through them.

Many use cases can be handled with short, specific filters. As a rule, prefer equality filters over substring filters.

DS servers reject unindexed searches by default, because unindexed searches are resource-intensive. If your application needs to use a filter that results in an unindexed search, work with the directory administrator to find a solution, such as adding the indexes required for your search filters.

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

Make Modifications Specific

Specific modifications help directory servers apply and replicate your changes more effectively.

When you modify attributes with multiple values, such as a static group member attribute, replace or delete specific values individually, rather than replacing the entire list of values.

Trust Result Codes

Trust the LDAP result code from the directory server. For example, if you request a modification, and you get a success result, consider the operation a success. Do not immediately issue a search to get the modified entry.

LDAP replication model is loosely convergent. In other words, the directory server sends you the success result *before* replicating the change to every directory server replica across the network. If you issue a read immediately after a write, a load balancer may direct the request to another replica. The result might differ from what you expect.

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

Handle Input Securely

When taking input directly from a user or another program, use appropriate methods to sanitize the data. Failure to sanitize the input data can leave your application vulnerable to injection attacks.

For Java applications, the Directory Services `format()` methods for filters and DNs are similar to the Java `String.format()` methods. In addition to formatting the output, they escape the input objects. When building a search filter, use one of the methods of the DS APIs to escape input.

Check Group Membership on the Account, Not the Group

Reading an entire large static group entry to check membership is wasteful.

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

Check Support for Features You Use

Directory servers expose their capabilities as operational attribute values on the root DSE, which is the entry whose DN is an empty string, `""`.

This lets your application discover capabilities at run time, rather than storing configuration separately. Putting effort into checking directory capabilities makes your application easier to deploy and to maintain.

For example, rather than hard-coding `dc=example,dc=com` as a base DN in your configuration, read the root DSE `namingContexts` attribute.

Directory servers also expose their schema over LDAP. The root DSE attribute `subschemaSubentry` shows the DN of the entry for LDAP schema definitions.

Store Large Attribute Values By Reference

To serve results quickly with high availability, directory servers cache content and replicate it everywhere. If you already store large attribute values elsewhere, such as photos or audio messages, keep only a reference to external content in a user’s account.

Take Care With Persistent Search and Server-Side Sorting

A persistent search lets your application receive updates from the server as they happen by keeping the connection open and forcing the server to check whether to return additional results any time it performs a modification in the scope of your search. Directory administrators therefore might hesitate to grant persistent search access to your application.

DS servers expose a change log to let you discover updates with less overhead. If you do have to use a persistent search instead, try to narrow the scope of your search.

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

Reuse Schemas Where Possible

DS servers come with schema definitions for a wide range of standard object classes and attribute types. Directories use unique, IANA-registered object identifiers (OIDs) to avoid object class and attribute type name clashes. The overall goal is Internet-wide interoperability.

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

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

When you have defined schema required for your application, work with the directory administrator to add your definitions to the directory service. DS servers let directory administrators update schema definitions over LDAP. There is no need to interrupt the service to add your application. Directory administrators can, however, have other reasons why they hesitate to add your schema definitions. Coming to the discussion prepared with good schema definitions, explanations of why they should be added, and evident regard for interoperability makes it easier for the directory administrator to grant your request.

Read Directory Server Schemas During Initialization

By default, Directory Services APIs use a minimal, built-in core schema, rather than reading the schema from the server. Doing so automatically would incur a significant performance cost. Unless schemas change, your application only needs to read them once.

When you start your application, read directory server schemas as a one-off initialization step.

Once you have the directory server schema definitions, use them to validate entries.

Handle Referrals

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

Troubleshooting: Check Result Codes

LDAP result codes are standard, and listed in LDAP Result Codes.

When your application receives a result, it must rely on the result code value to determine what action to take. When the result is not what you expect, read or at least log the additional message information.

Troubleshooting: Check Server Log Files

If you can read the directory server access log, then check what the server did with your application's request. The following excerpt shows a successful search by `cn=My App,ou=Apps,dc=example,dc=com`:

▼ Show excerpt

```
{"eventName":"DJ-LDAP","client":
{"ip":"127.0.0.1","port":59891},"server":
{"ip":"127.0.0.1","port":1389},"request":
{"protocol":"LDAP","operation":"CONNECT","connId":0},"transactio
nId":"0","response":
{"status":"SUCCESSFUL","statusCode":"0","elapsedTime":0,"elapsed
TimeUnits":"MILLISECONDS"},"timestamp":"2016-10-
20T15:48:36.933Z","_id":"11d5fdaf-79ac-4677-a640-805db1c35af0-
3"}
{"eventName":"DJ-LDAP","client":
{"ip":"127.0.0.1","port":59891},"server":
{"ip":"127.0.0.1","port":1389},"request":
{"protocol":"LDAP","operation":"EXTENDED","connId":0,"msgId":1,"
name":"StartTLS","oid":"1.3.6.1.4.1.1466.20037"},"transactionId"
:"0","response":
{"status":"SUCCESSFUL","statusCode":"0","elapsedTime":3,"elapsed
TimeUnits":"MILLISECONDS"},"timestamp":"2016-10-
20T15:48:36.945Z","_id":"11d5fdaf-79ac-4677-a640-805db1c35af0-
5"}
{"eventName":"DJ-LDAP","client":
{"ip":"127.0.0.1","port":59891},"server":
{"ip":"127.0.0.1","port":1389},"request":
{"protocol":"LDAP","operation":"BIND","connId":0,"msgId":2,"vers
ion":"3","authType":"Simple","dn":"cn=My
App,ou=Apps,dc=example,dc=com"},"transactionId":"0","response":
{"status":"SUCCESSFUL","statusCode":"0","elapsedTime":6,"elapsed
TimeUnits":"MILLISECONDS"},"userId":"cn=My
App,ou=Apps,dc=example,dc=com","timestamp":"2016-10-
20T15:48:38.462Z","_id":"11d5fdaf-79ac-4677-a640-805db1c35af0-
7"}
{"eventName":"DJ-LDAP","client":
{"ip":"127.0.0.1","port":59891},"server":
{"ip":"127.0.0.1","port":1389},"request":
{"protocol":"LDAP","operation":"SEARCH","connId":0,"msgId":3,"dn
":"dc=example,dc=com","scope":"sub","filter":"
(uid=kvaughan)","attrs":
["isMemberOf"]},"transactionId":"0","response":
{"status":"SUCCESSFUL","statusCode":"0","elapsedTime":6,"elapsed
TimeUnits":"MILLISECONDS","nentries":1},"timestamp":"2016-10-
20T15:48:38.472Z","_id":"11d5fdaf-79ac-4677-a640-805db1c35af0-
9"}
{"eventName":"DJ-LDAP","client":
{"ip":"127.0.0.1","port":59891},"server":
{"ip":"127.0.0.1","port":1389},"request":
{"protocol":"LDAP","operation":"UNBIND","connId":0,"msgId":4},"t
ransactionId":"0","timestamp":"2016-10-
20T15:48:38.480Z","_id":"11d5fdaf-79ac-4677-a640-805db1c35af0-
11"}
{"eventName":"DJ-LDAP","client":
{"ip":"127.0.0.1","port":59891},"server":
{"ip":"127.0.0.1","port":1389},"request":
{"protocol":"LDAP","operation":"DISCONNECT","connId":0},"transac
tionId":"0","response":
{"status":"SUCCESSFUL","statusCode":"0","elapsedTime":0,"elapsed
TimeUnits":"MILLISECONDS","reason":"Client
Unbind"},"timestamp":"2016-10-20T15:48:38.481Z","_id":"11d5fdaf-
79ac-4677-a640-805db1c35af0-13"}
```

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

Troubleshooting: Inspect Network Traffic

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

Next Steps

Once you have worked through the examples in this guide, try the following suggestions:

Learn About Replication

Data replication is sometimes called the "killer feature" of LDAP directories. Its strengths are in enabling very high availability for directory services even during network outages, and automatically resolving conflicts that can occur when the network is down, for example. LDAP directories have been improving and hardening replication features for decades.

Its weaknesses are that replication protocols have not been standardized for interoperability, and that unwary developers can misunderstand its property of eventual consistency if they are too used to the strong, immediate consistency of monolithic, transactional databases.

Replication necessarily involves multiple servers and additional configuration. You can learn more about it by reading Replication, and trying the examples in that page.

Browse DS Documentation

Category	Topics Covered
Release notes	DS features, fixes, and known issues
Deployment	Deploying Directory Services in on-premises and cloud environments
Installation	Installing DS software
Upgrade	Upgrading DS software
Configuration	Configuring DS servers after installation
Security	Ensuring a Directory Services deployment is secure
Maintenance	Day-to-day operations for maintaining DS servers

Category	Topics Covered
Logging	Configuring DS server logs
Monitoring	What to monitor when running DS servers, and where to look for metrics and other information
Use LDAP	How to use LDAP features and command-line tools
Use REST/HTTP	How to configure and use DS REST APIs for HTTP access
Configuration Reference	The **`dsconfig`** subcommands and server configuration properties
DS Javadoc	Evolving LDAP SDK and server APIs, including ForgeRock common APIs
LDAP Reference	LDAP-specific features of DS software
LDAP Schema Reference	All default LDAP schema, including monitoring attributes and object classes
Log Reference	DS server error log messages by category and ID
Tools Reference	Tools bundled with DS software

Try Third-Party Tools

LDAP is a standard protocol, and so you can use LDAP-compliant third-party tools to manage directory data:

- Admin4
- Apache Directory Studio
- JXplorer and JXWorkBench
- phpLDAPadmin
- Softerra LDAP Administrator
- web2ldap

Many software solutions include support for LDAP authentication and LDAP-based address books.

ForgeRock does not endorse or support third-party tools.

Use DS With AM

- Back End Directory Servers in the AM *Deployment Planning Guide*
- Preparing External Stores in the AM *Installation Guide*
- Configuring External CTS Token Stores in the AM *Core Token Service Guide*
- You can install DS directory servers for use as external AM stores.

  For details, see Setup Profiles.

Use DS With IDM

- External DS Repository and Select a Repository in the IDM *Installation Guide*

  Also see Install DS as an IDM Repository.
- One Way Synchronization From LDAP to IDM, Two Way Synchronization Between LDAP and IDM, and other LDAP-related pages in the IDM *Samples Guide*
- DS Repository Configuration and Generic and Explicit Mappings With a DS Repository in the IDM *Object Modeling Guide*
- Synchronizing Passwords With ForgeRock Directory Services (DS) in the IDM *Password Synchronization Plugin Guide*

Remove DS Software

For details, see Uninstallation.

Glossary

Abandon operation
LDAP operation to stop processing of a request in progress, after which the server drops the connection without a reply to the client application.

Access control
Control to grant or to deny access to a resource.

Access control instruction (ACI)
Instruction added as a directory entry attribute for fine-grained control over what a given user or group member is authorized to do in terms of LDAP operations and access to user data.

ACIs are implemented independently from privileges, which apply to administrative operations.

See also: Privilege

Access control list (ACL)

An access control list connects a user or group of users to one or more security entitlements. For example, users in group sales are granted the entitlement read-only to some financial data.

Access log
Server log tracing the operations the server processes including timestamps, connection information, and information about the operation itself.

Account lockout
The act of making an account temporarily or permanently inactive after successive authentication failures.

Active user
A user that has the ability to authenticate and use the services, having valid credentials.

Add operation
LDAP operation to add a new entry or entries to the directory.

Anonymous
A user that does not need to authenticate, and is unknown to the system.

Anonymous bind
A bind operation using simple authentication with an empty DN and an empty password, allowing anonymous access such as reading public information.

Approximate index
Index is used to match values that "sound like" those provided in the filter.

Attribute
Properties of a directory entry, stored as one or more key-value pairs. Typical examples include the common name (`cn`) to store the user’s full name and variations of the name, user ID (`uid`) to store a unique identifier for the entry, and `mail` to store email addresses.

Audit log
Type of access log that dumps changes in LDIF.

Authentication
The process of verifying who is requesting access to a resource; the act of confirming the identity of a principal.

Authorization
The process of determining whether access should be granted to an individual based on information about that individual; the act of determining whether to grant or to deny a principal access to a resource.

Backend

Repository that stores directory data. Different implementations with different capabilities exist.

Binary copy
Backup files from one replica are restored on another replica.

Bind operation
LDAP authentication operation to determine the client’s identity in LDAP terms, the identity which is later used by the server to authorize (or not) access to directory data that the client wants to lookup or change.

Branch
The distinguished name (DN) of a non-leaf entry in the Directory Information Tree (DIT), and that entry and all its subordinates taken together.

Some administrative operations allow you to include or exclude branches by specifying the DN of the branch.

See also: Suffix

Collective attribute
A standard mechanism for defining attributes that appear on all the entries in a particular subtree.

Compare operation
LDAP operation to compare a specified attribute value with the value stored on an entry in the directory.

Control
Information added to an LDAP message to further specify how an LDAP operation should be processed. DS supports many LDAP controls.

Change sequence number (CSN)
An opaque string uniquely identifying a single change to directory data. A CSN indicates exactly when a change occurred on which replica. An example CSN is `010f016df804edca0000008fevaluation-only`.

DS replication uses CSNs to replay replicated operations consistently on all replicas. DS replicas record CSNs in historical data values for `ds-sync-state` and `ds-sync-hist` attributes.

When troubleshooting replication data consistency, it can be useful to interpret CSNs. For details, see the ForgeRock Knowledge Base.

Database cache
Memory space set aside to hold database content.

Debug log
Server log tracing details needed to troubleshoot a problem in the server.

Delete operation

LDAP operation to remove an existing entry or entries from the directory.

Directory

A directory is a network service which lists participants in the network such as users, computers, printers, and groups. The directory provides a convenient, centralized, and robust mechanism for publishing and consuming information about network participants.

Directory hierarchy

A directory can be organized into a hierarchy in order to make it easier to browse or manage. Directory hierarchies normally represent something in the physical world, such as organizational hierarchies or physical locations.

For example, the top level of a directory may represent a company, the next level down divisions, the next level down departments, and down the hierarchy. Alternately, the top level may represent the world, the next level down countries, next states or provinces, and next cities.

Directory Information Tree (DIT)

A set of directory entries organized hierarchically in a tree structure, where the vertices are the entries, and the arcs between vertices define relationships between entries.

Directory object

A directory object is an item in a directory. Example objects include users, user groups, computers, and more. Objects may be organized into a hierarchy and contain identifying attributes.

See also: Entry

Directory proxy server

Server that forwards LDAP requests to remote directory servers. A standalone directory proxy server does not store user data.

Directory server

Server application for centralizing information about network participants. A highly available directory service consists of multiple directory servers configured to replicate directory data.

See also: Replication

Directory Services Markup Language (DSML)

Standard language to access directory services using XML. DMSL v1 defined an XML mapping of LDAP objects, while DSMLv2 maps the LDAP Protocol and data model to XML.

Directory superuser

Directory account with privileges to do full administration of the DS server, including bypassing access control evaluation, changing access controls, and changing administrative privileges.

See also: Superuser

Distinguished name (DN)
Fully qualified name for a directory entry, such as `uid=bjensen,ou=People,dc=example,dc=com`, built by concatenating the entry RDN (`uid=bjensen`) with the DN of the parent entry (`ou=People,dc=example,dc=com`).

Domain
A replication domain consists of several directory servers sharing the same synchronized set of data.

The base DN of a replication domain specifies the base DN of the replicated data.

DSML gateway
Standalone web application that translates DSML requests from client applications to LDAP requests to a directory service, and LDAP responses from a directory service to DSML responses to client applications.

Dynamic group
Group that specifies members using LDAP URLs.

Entry
An entry is an object in the directory, defined by one of more object classes, and their related attributes.

Entry cache
Memory space set aside to hold frequently accessed, large entries, such as static groups.

Equality index
Index used to match values that correspond exactly (though generally without case sensitivity) to the value provided in the search filter.

Errors log
Server log tracing server events, error conditions, and warnings, categorized and identified by severity.

Etime
Elapsed time within the server to process a request, starting from the moment the decoded operation is available to be processed by a worker thread.

Export
Save directory data in an LDIF file.

Extended operation

Additional LDAP operation not included in the original standards. DS servers support several standard LDAP extended operations.

Extensible match index

Index for a matching rule other than approximate, equality, ordering, presence, substring or VLV, such as an index for generalized time.

External user

An individual that accesses company resources or services but is not working for the company. Typically, a customer or partner.

Filter

An LDAP search filter is an expression that the server uses to find entries that match a search request, such as `(mail=*@example.com)` to match all entries having an email address in the example.com domain.

Group

Entry identifying a set of members whose entries are also in the directory.

Generation ID

The initial state identifier for a replicated directory server base DN. It is a hash of the first 1000 entries of the base DN, computed when creating the backend, importing data from LDIF, or initializing replication.

Replication can only proceed between base DNs that have the same generation ID.

Idle time limit

Defines how long DS allows idle connections to remain open.

Import

Read in and index directory data from an LDIF file.

Inactive user

An entry in the directory that once represented a user but which is now no longer able to be authenticated.

Index

Directory server backend feature to allow quick lookup of entries based on their attribute values.

See also: Approximate index, Equality index, Extensible match index, Ordering index, Presence index, Substring index, VLV index, Index entry limit

Index entry limit

When the number of entries that an index key points to exceeds the index entry limit, DS stops maintaining the list of entries for that index key.

Internal user

An individual who works within the company either as an employee or as a contractor.

LDAP Data Interchange Format (LDIF)
Standard, portable, text-based representation of directory content.

See RFC 2849.

LDAP URL
LDAP Uniform Resource Locator, such as `ldaps://ds.example.com:636/dc=example,dc=com??sub?(uid=bjensen)`.

See RFC 2255.

LDAPS
LDAP over SSL.

Lightweight Directory Access Protocol (LDAP)
A simple and standardized network protocol used by applications to connect to a directory, search for objects and add, edit or remove objects.

See RFC 4510.

Lookthrough limit
Defines the maximum number of candidate entries DS considers when processing a search.

Matching rule
Defines rules for performing matching operations against assertion values. Matching rules are frequently associated with an attribute syntax, and are used to compare values according to that syntax.

For example, the `distinguishedNameEqualityMatch` matching rule can be used to determine whether two DNs are equal and can ignore unnecessary spaces around commas and equal signs, differences in capitalization in attribute names, and other discrepancies.

Modify DN operation
LDAP modification operation to request that the server change the distinguished name of an entry.

Modify operation
LDAP modification operation to request that the server change one or more attributes of an entry.

Naming context
Base DN under which client applications can look for user data.

Object class

Identifies entries that share certain characteristics. Most commonly, an entry's object classes define the attributes that must and may be present on the entry.

Object classes are stored on entries as values of the `objectClass` attribute. Object classes are defined in the directory schema, and can be *abstract* (defining characteristics for other object classes to inherit), *structural* (defining the basic structure of an entry, one structural inheritance per entry), or *auxiliary* (for decorating entries already having a structural object class with other required and optional attributes).

Object identifier (OID)
String that uniquely identifies an object, such as `0.9.2342.19200300.100.1.1` for the user ID attribute or `1.3.6.1.4.1.1466.115.121.1.15` for `DirectoryString` syntax.

Operational attribute
An attribute that has a special (operational) meaning for the server, such as `pwdPolicySubentry` or `modifyTimestamp`.

Ordering index
Index used to match values for a filter that specifies a range.

Password policy
A set of rules regarding what sequence of characters constitutes an acceptable password. Acceptable passwords are generally those that would be too difficult for another user, or an automated program to guess and thereby defeat the password mechanism.

Password policies may require a minimum length, a mixture of different types of characters (lowercase, uppercase, digits, punctuation marks, and other characters), avoiding dictionary words or passwords based on the user's name, and other attributes.

Password policies may also require that users not reuse old passwords and that users change their passwords regularly.

Password reset
Password change performed by a user other than the user who owns the entry.

Password storage scheme
Mechanism for encoding user passwords stored on directory entries. DS implements a number of password storage schemes.

Password validator
Mechanism for determining whether a proposed password is acceptable for use. DS implements a number of password validators.

Plugin

Java library with accompanying configuration that implements a feature through processing that is not essential to the core operation of DS servers.

As the name indicates, plugins can be plugged in to an installed server for immediate configuration and use without recompiling the server.

DS servers invoke plugins at specific points in the lifecycle of a client request. The DS configuration framework lets directory administrators manage plugins with the same tools used to manage the server.

Presence index
Index used to match the fact that an attribute is present on the entry, regardless of the value.

Principal
Entity that can be authenticated, such as a user, a device, or an application.

Privilege
Server configuration settings controlling access to administrative operations such as exporting and importing data, restarting the server, performing password reset, and changing the server configuration.

Privileges are implemented independently from access control instructions (ACI), which apply to LDAP operations and user data.

See also: Access control instruction

Referential integrity
Ensuring that group membership remains consistent following changes to member entries.

Referint log
Server log tracing referential integrity events, with entries similar to the errors log.

Referral
Reference to another directory location, which can be another directory server running elsewhere or another container on the same server, where the current operation can be processed.

Relative distinguished name (RDN)
Initial portion of a DN that distinguishes the entry from all other entries at the same level, such as `uid=bjensen in uid=bjensen,ou=People,dc=example,dc=com`.

Replica
Directory server this is configured to use replication.

Replication
Data synchronization that ensures all directory servers participating eventually share a consistent set of directory data.

Replication log

Server log tracing replication events, with entries similar to the errors log.

Replication server

Server dedicated to transmitting replication messages. A standalone replication server does not store user data.

REST to LDAP gateway

Standalone web application that translates RESTful HTTP requests from client applications to LDAP requests to directory services, and LDAP responses from directory services to HTTP responses to client applications.

Root DSE

The directory entry with distinguished name "" (empty string), where DSE is an acronym for *DSA-Specific Entry*. DSA is an acronym for *Directory Server Agent*, a single directory server.

The root DSE serves to expose information over LDAP about what the directory server supports in terms of LDAP controls, auth password schemes, SASL mechanisms, LDAP protocol versions, naming contexts, features, LDAP extended operations, and other information.

Schema

LDAP schema defines the object classes, attributes types, attribute value syntaxes, matching rules and other constrains on entries held by the directory server.

Search filter

See: Filter

Search operation

LDAP lookup operation where a client requests that the server return entries based on an LDAP filter, and a base DN under which to search.

Simple authentication

Bind operation performed with a user's entry DN and user's password.

Use simple authentication only if the network connection is secure.

Size limit

Sets the maximum number of entries returned for a search.

Static group

Group that enumerates member entries.

Subentry

An entry, such as a password policy entry, that resides with the user data but holds operational data, and is not visible in search results unless explicitly requested.

Substring index

Index used to match values specified with wildcards in the filter.

Suffix
The distinguished name (DN) of a root entry in the Directory Information Tree (DIT), and that entry and all its subordinates taken together as a single object of administrative tasks such as export, import, indexing, and replication.

Superuser
User with privileges to perform unconstrained administrative actions on DS server. This account is analogous to the UNIX `root` and Windows `Administrator` accounts.

The conventional default superuser DN is `uid=admin` . You can create additional superuser accounts, each with different administrative privileges.

Superuser privileges include the following:

- `bypass-acl` : The holder is not subject to access control.
- `privilege-change` : The holder can edit administrative privileges.
- `proxied-auth` : The holder can make requests on behalf of another user, including directory superusers.

See also: Directory superuser, Privilege

Task
Mechanism to provide remote access to server administrative functions.

DS software supports tasks to back up and restore backends, to import and export LDIF files, and to stop and restart the server.

Time limit
Defines the maximum processing time DS devotes to a search operation.

Unbind operation
LDAP operation to release resources at the end of a session.

Unindexed search
Search operation for which no matching index is available.

If no indexes are applicable, then the directory server potentially has to go through all entries to look for candidate matches. For this reason, the `unindexed-search` privilege, which allows users to request searches for which no applicable index exists, is reserved for the directory manager by default.

User
An entry that represents an individual that can be authenticated through credentials contained or referenced by its attributes. A user may represent an internal user or an external user, and may be an active user or an inactive user.

User attribute
: An attribute for storing user data on a directory entry such as `mail` or `givenname`.

Virtual attribute
: An attribute with dynamically generated values that appear in entries but are not persistently stored in the backend.

Virtual directory
: An application that exposes a consolidated view of multiple physical directories over an LDAP interface. Consumers of the directory information connect to the virtual directory’s LDAP service.

  Behind the scenes, requests for information and updates to the directory are sent to one or more physical directories where the actual information resides. Virtual directories enable organizations to create a consolidated view of information that for legal or technical reasons cannot be consolidated into a single physical copy.

Virtual list view (VLV) index
: Browsing index designed to help the directory server respond to client applications that need, for example, to browse through a long list of results a page at a time in a GUI.

Virtual static group
: DS group that lets applications see dynamic groups as what appear to be static groups.

X.500
: A family of standardized protocols for accessing, browsing and maintaining a directory. X.500 is functionally similar to LDAP, but is generally considered to be more complex, and has consequently not been widely adopted.

Was this helpful?